[redacted]

IMAGE UNDERSTANDING RESEARCH AND ITS APPLICATION TO CARTOGRAPHY AND COMPUTER-BASED ANALYSIS OF AERIAL IMAGERY

Sixth and Seventh Semiannual Technical Reports
Covering the Period April 1, 1982 to March 31, 1983

Contract Amount:	$2,668,395
Effective Date:	September 5, 1979
Expiration Date:	June 10, 1983

May 1983

By: Martin A. Fischler, Program Director
Principal Investigator, (415)859-5106

Artificial Intelligence Center
Computer Science and Technology Division

Prepared for:

Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

Contract No. MDA903-79-C-0588
DARPA Order No. 3862
Program Code No. 61101E
SRI Project 1009

SRI International
333 Ravenswood Avenue
Menlo Park, California 94025
(415) 326-6200
Cable: SRI INTL MPK
TWX: 910-373-2046

SRI International®

[redacted]

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Sixth and Seventh Semiannual Repts	2 GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Image Understanding Research and Its Application to Cartography and Computer-Based Analysis of Aerial Imagery		5. TYPE OF REPORT & PERIOD COVERED Combined Semiannual Technical 4/1-9/30/82 & 10/1/82-3/31/83 6. PERFORMING ORG. REPORT NUMBER 1009 6th & 7th Semiannual
7. AUTHOR(s) Martin A. Fischler		8. CONTRACT OR GRANT NUMBER(s) MDA903-79-C-0588
9. PERFORMING ORGANIZATION NAME AND ADDRESS SRI International 333 Ravenswood Avenue Menlo Park, California 94025		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61101E
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, Virginia 22209		12. REPORT DATE May 1983 13. NUMBER OF PAGES 155
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) DCASMA, San Francisco 1250 Bayhill Drive San Bruno, California 94066		15. SECURITY CLASS (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Image Understanding, Computer Vision, DARPA/DMA Testbed		

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Our principal objective in this research program has been to obtain solutions to fundamental problems in computer vision that have broad military relevance, particularly in the areas of cartography and photo interpretation. Now completed research has been directed towards developing automated techniques for stereo-compilation, delineation of linear features, scene partitioning, image matching, and image to database correspondence.

(CONTINUED ON NEXT PAGE)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT (continued)

In addition to our own research, we have designed and implemented an integrated testbed system that incorporates results of research produced throughout the Image Understanding community. This system provides a coherent framework for demonstration and evaluation of the accomplishments of DARPA's Image Understanding program, thereby facilitating transfer of this technology to appropriate military organizations.

CONTENTS

PREFACE

This report combines the semiannual technical reports for the periods April 1, 1982 to September 30, 1982 and October 1, 1982 to March 31, 1983.

ACKNOWLEDGMENT

Contributors to the SRI Image Understanding Program include the following staff members: S. T. Barnard, R. C. Bolles, M. A. Fischler, A. J. Hanson, D. L. Kashtan, K. Laws, A. P. Pentland, L. H. Quam, G. Smith, and H. C. Wolf.

I INTRODUCTION

Research at SRI International under the DARPA Image Understanding Program was initiated to investigate ways in which diverse sources of knowledge might be brought to bear on the problem of analyzing and interpreting aerial images. An initial, exploratory phase of research identified various means for exploiting stored knowledge in the processing of aerial photographs for such military applications as cartography, intelligence, weapon guidance, and targeting. A key concept was the use of a generalized digital map to guide the process of image analysis. The results of this earlier work were integrated into an interactive computer system called "Hawkeye" [1].

Research subsequently focused on development of a program capable of expert performance in a specific task domain--road monitoring. The primary objective of this work was to build a computer system (called the Road Expert) that "understands" the nature of roads and road events; it is capable of performing such tasks as:

* Finding roads in aerial imagery.
* Distinguishing vehicles on roads from shadows, signposts, road markings, etc.
* Comparing multiple images and symbolic information pertaining to the same road segment, and deciding whether significant changes have occurred.

The general approach, and technical details of the Road Expert's components, are contained in the references [2-8]. We have integrated some of these separate components into a system that facilitates testing and evaluation and have transferred this system to the DARPA/DMA Testbed.

We are now close to the completion of this contract and are in the final stages of two remaining major activities:

We have completed our planned program of machine vision research--specifically, selected problems in the areas of three-dimensional terrain understanding, linear-feature analysis, image partitioning, and image description and matching. This research program has been centered on the concept that image interpretation, except in the simplest situations, involves a form of reasoning ("perceptual reasoning") that is characterized by the need to integrate information from multiple sources, which are typically incommensurate and often erroneous or in conflict. We have developed a number of new techniques, and even complete paradigms, for effecting the knowledge-integration task. These new techniques have been incorporated in the more focused efforts discussed in Section II, which address significant problems in scene analysis.

We have almost completed work on the DARPA/DMA Testbed project; the main items remaining are primarily concerned with documentation. In Section III of this report we describe the current status of the

Testbed. (Hanson and Fischler [9] describe the purpose and goals of the Testbed project in greater detail.)

II RESEARCH PROGRESS AND ACCOMPLISHMENTS

A. Three-Dimensional Compilation and Interpretation

The problem of stereo reconstruction is almost synonymous with the problem of machine vision--use of imaged data to (geometrically) model a sensed scene. A key concept in our approach [10-13] is the use of global physical and semantic constraints (e.g., sun location, vanishing points, edge detection and classification, skyline delineation, etc.) to resolve local ambiguities that defeat conventional stereo-matching techniques in mapping cultural or urban scenes, i.e., scenes that contain featureless areas and large numbers of occlusion edges, or scenes that are represented by widely separated or oblique views.

When a stereo pair of images are matched, even with the best possible use of available data (because of some of the problems mentioned above, e.g., occlusion and featureless areas), we generally can do no better than to compute a sparse depth map of the imaged scene. However, for many tasks, a sparse depth map is inadequate. We want a complete model, which accurately portrays the scene's surfaces. To achieve this goal, we must obtain the missing surface shape information from the shading of the images of the stereo pair. We have made significant progress in understanding what is possible with respect to

surface interpolation using scene shading. Pentland [14] and Smith [15] discuss some of our recent work in this research area.

B. Detection, Delineation, and Interpretation of Linear Features in Aerial Imagery

We have developed a system, called the "Road Expert," which can precisely delineate roads in both high- and low-resolution aerial imagery, and classify the visible objects that fall within the road boundaries [2-8]. A demonstration version of the Road Expert has been installed on the DARPA/DMA testbed. We have investigated extensions of the above work to the problem of delineating other types of linear structures, such as rivers [16], and have recently made a significant advance towards developing a completely autonomous system for delineation of arbitrary linear structures [17].

C. Image Matching and Image-to-Database Correspondence

We have developed a new paradigm, called Random Sample Consensus (RANSAC), for fitting a model to data containing a significant percentage of gross errors, and have applied this paradigm to the solution of the matching/correspondence problem [18]. A RANSAC-based camera model solver has been developed and installed on the testbed. We expect that RANSAC will be equally applicable to a wide range of other model-based interpretation tasks, and, under a separate contract, are investigating its use for recognizing and labeling known two- and three-

dimensional scene features, even though seen under unusual viewing or illumination conditions and even when the objects are partially occluded.

D. Image Partitioning, Intensity Modeling, and Material Identification

Our goal in this effort is the development of techniques for partitioning and modeling the material composition of a scene from available imagery. In order to recover information about actual surface reflectances and physical composition, the problem of intensity modeling must be addressed. We have developed methods for deriving absolute scene-intensity information without calibration data (such as a step wedge exposed on the image) based on knowing the identity of the material composition of the surfaces at a few points in the image--this capability is required to partition the image into labeled regions of given material types [10].

III STATUS OF THE DARPA/DMA IMAGE UNDERSTANDING TESTBED

The DARPA/DMA Image Understanding Testbed established at SRI as part of the DARPA Image Understanding research program constitutes a coherent body of software running in a standard hardware environment. Demonstrations of the features and capabilities of all IU-community-contributed software are available; detailed evaluations have been carried out for selected modules. The Testbed is now established as a technology transfer tool, which can be utilized by appropriate agencies to evaluate the applicability of the contributed scene analysis techniques.

Documentation of the Testbed is entering its final phase. Final drafts of the User's manual, the Programmer's manual, and the System Manager's manual are available and will soon pass through the required editing and approval procedures. The evaluation reports for the Ghough and Phoenix programs have been completed and are included as appendices to this report. Work is almost finished on the evaluation report for the Relaxation package and on user-level documentation of those contributions for which no detailed evaluation is planned. More extensive studies of the various approaches to stereo compilation now available on the Testbed will be integrated into a separate, ongoing, DARPA-supported research effort.

The Testbed is now sufficiently well-defined that exact copies of the entire system can be configured, if desired. SRI, under a separate contract, is just completing the installation of a Testbed copy (hardware and software) at the U.S. Army Engineer Topographic Laboratories (ETL) at Fort Belvoir. A Lisp Machine will be added to the ETL configuration later in the year. SRI will also be supplying Lisp Machines and Lisp Machine software to the DMAHTC and DMAAC branches of the Defense Mapping Agency. SRI has been closely involved in efforts to ensure that the upgrade of the DMA AFES/RWPF facilities to the VAX-11/780 CPU can incorporate the Image Understanding Testbed capabilities, as well as supporting the Lisp Machines.

The Testbed software system and its utilities are being prepared for export to university researchers in the IU program as well as to other U.S. Government agencies interested in establishing Tested copies. SRI has developed a simple license agreement to help protect Testbed contributors and restrict use of the software to appropriate academic and government research environments.

REFERENCES

1. H.G. Barrow, et al., "Interactive Aids for Cartography and Photo Interpretation: Progress Report, October 1977," in Proceedings: Image Understanding Workshop, pp. 111-127 (October 1977).

2. M.A. Fischler, et al., "Interactive Aids for Cartography and Photo Interpretation," Semiannual Technical Report, SRI Project 5300, SRI International, Menlo Park, California (October 1978 and May 1979).

3. L. Quam, "Road Tracking and Anomaly Detection," in Proceedings: Image Understanding Workshop>, pp. 51-55 (May 1978).

4. R.C. Bolles, et al., "The SRI Road Expert: Image-to-Database Correspondence," in Proceedings: Image Understanding Workshop, pp. 163-174 (November 1978).

5. G.J. Agin, "Knowledge-Based Detection and Classification of Vehicles and Other Objects in Aerial Road Images," in Proceedings: Image Understanding Workshop, pp. 66-71 (April 1979).

6. R.C. Bolles, et al., "Automatic Determination of Image-to-Database Correspondence," in Proceedings Sixth IJCAI (1979).

7. M.A. Fischler, "The SRI Image Understanding Program," in Proceedings: Image Understanding Workshop (November 1979).

8. M.A. Fischler, J.M. Tenenbaum, and H.C. Wolf, "Detection of Roads and Linear Structures in Low-Resolution Aerial Imagery Using a Multisource Knowledge IntegrationeTechnique," Computer Graphics and Image Processing, Vol. 15(3), pp. 201-223 (March 1981).

9. A.J. Hanson and M.A. Fischler, "The DARPA/DMA Image Understanding Testbed," Proceedings: Image Understanding Workshop (September 1982).

10. M.A. Fischler, et.al., "Modeling and Using Physical Constraints in Scene Analysis," AAAI-82 (August 1982).

11. S. Barnard and M.A. Fischler, "Computational Stereo," ACM Computing Surveys, Vol. 14(4) (December 1982).

12. S. Barnard, "Methods for Interpreting Perspective Images," AI Journal (in press, 1983).

13. A. Pentland, "Depth of Scene from Depth of Field," Proceedings of the Image Understanding Workshop (September 1982).

14. A. Pentland, "Local Analysis of the Image: Limitations and Uses of Shading," Proceedings of the (IEEE) Workshop on Computer Vision: Representation and Control, Rindge, New Hampshire (August 1982).

15. G.B. Smith, "The Recovery of Surface Orientation From Image Irradiance," Proceedings of the Image Understanding Workshop (September 1982).

16. G.B. Smith, "Detection of Rivers in Low-Resolution Aerial Imagery," SRI Technical Note 244 (June 1981).

17. M.A. Fischler and H.C. Wolf "Linear Delineation," " Proceedings of the Image Understanding Workshop (September 1982). Also, will be presented at IEEE CVPR-83 (June 1983).

18. M.A. Fischler and R.C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," CACM, Vol. 24(6), pp. 381-395 (June 1981).

Appendix A

The Ghough Generalized Hough Transform Package: Description and Evaluation

THE GHOUGH GENERALIZED HOUGH TRANSFORM PACKAGE: DESCRIPTION AND EVALUATION

Technical Note

December 1982

By: Kenneth I. Laws, Computer Scientist

Artificial Intelligence Center
Computer Science and Technology Division

Program Development by:

Dana H. Ballard
Kenneth R. Sloan, Jr.
Bill Lampeter

University of Rochester

SRI Project 1009

The work reported herein was supported by the Defense Advanced Research Projects Agency under Contract No. MDA903-79-C-0588.

Foreword

The primary purpose of the Image Understanding (IU) Testbed is to provide a means for transferring technology from the DARPA-sponsored IU research program to DMA and to other organizations in the defense community.

The approach taken to achieve this purpose has two components:

(1) The establishment of a uniform environment as compatible as practical with the environments of research centers at universities participating in the IU research program. Thus, organizations obtaining copies of the Testbed can receive a continuing flow of new results derived from on-going research.

(2) The acquisition, integration, testing, and evaluation of selected scene analysis techniques that represent mature examples of generic areas of research activity. These contributions from participants in the IU research program will allow organizations with Testbed copies to begin the immediate exploration of applications of IU technology to problems in automated cartography and other areas of scene analysis.

The IU Testbed project was carried out under DARPA contract No. MDA903-79-C-0599. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the United States Government.

This report describes the GHOUGH generalized Hough transform package contributed by the University of Rochester and provides an evaluation of its features.

Andrew J. Hanson
Testbed Coordinator
Artificial Intelligence Center
SRI International

Abstract

GHOUGH is a computer program for detecting instances of a given shape within an image. It may be used for cueing, counting, or mensuration. GHOUGH can find instances that are displaced, rescaled, rotated, or incomplete relative to the shape template. They are detected by computing a "generalized Hough transform" of the image edge elements. Each edge element votes for all those instances of the shape that could contain it; the votes are tallied and the best supported instances are reported as likely matches.

GHOUGH was contributed to the DARPA Image Understanding Testbed at SRI by the University of Rochester. This report summarizes applications for which GHOUGH is suited, the history and nature of the algorithm, details of the Testbed implementation, the manner in which GHOUGH is invoked and controlled, the types of results that can be expected, and suggestions for further development. The scientific contributions of this technical note are the analysis of GHOUGH's parameter settings and performance characteristics.

Table of Contents

Section 1

Introduction

GHOUGH is a computer program for detecting instances of a given shape within an image. The instances may be displaced, rescaled, rotated, or incomplete relative to the shape template. Shape instances are detected by computing a "generalized Hough transform" of the image edge elements. Each edge element votes for all those instances of the shape that could contain the observed edge. The votes are tallied and the best supported instances are reported as likely matches.

The package was originally written in the SAIL language by Drs. Dana H. Ballard and Kenneth R. Sloan, Jr., at the University of Rochester. It was rewritten in C and prepared for delivery to the IU Testbed by Bill Lampeter at Rochester. At SRI the code was integrated with utility and image display software derived from other contributions, particularly those from Carnegie-Mellon University. This integration was done by Dr. Kenneth I. Laws.

Numerous changes were made in the user interface to facilitate evaluation activities. A few extensions were made relating to multiple target detections, but the central algorithm has been left virtually unchanged. The information in this document should thus be considered supplementary to the material cited in the references.

This document includes both a user's guide to the GHOUGH generalized Hough transform package and an evaluation of the algorithm. Section 2 explains the nature and uses of the algorithm and suggests alternate approaches to similar data analysis problems. Section 3 surveys the historical development of generalized Hough techniques and presents the current algorithm in detail. Section 4 instructs the user in the mechanics of using the GHOUGH software. Section 5 goes into greater detail on the meaning of the user-specified parameters, documents the performance that may be expected in various circumstances, and presents the results of evaluation tests. Section 6 outlines a number of suggestions for improving the algorithm and its implementation.

Section 2

Background

This section presents a management view of the GHOUGH program. The template matching algorithm is briefly sketched. Typical applications and potential applications requiring further development of the algorithm are discussed, and related applications for which other algorithms are better suited are noted.

2.1. Method

GHOUGH is a program for detecting instances of a given shape within a digital image. The shape is described by a template -- a series of dots outlining the shape and forming a closed boundary. Only silhouette information is used: this provides noise immunity in many applications, but does limit the use of GHOUGH in infrared target identification and other low-resolution applications.

Templates are abstract shapes, without size or orientation. They are also stored without contrast information so that instances may be located against darker or lighter backgrounds. (For efficiency the GHOUGH program allows the user to restrict the search to particular sizes, orientations, contrast directions, or subareas in the image. Shape instances outside these ranges will not be found.)

The GHOUGH program passes an edge detector over the image. The edge detector computes both a gradient magnitude and a gradient direction at each point (pixel) in the image. The user must set a threshold value for choosing which gradients are strong enough to be considered valid edges.

Each edge is allowed to vote for all shape instances of which it could be a part. One vote is recorded for each point in the template at each permissible orientation and size. This vote corresponds to the computed (x, y) coordinate position of the shape center. (The shape position is indicated by an arbitrary reference point in the template. Typically this point is at the center of the shape, but for a corner template the logical position would be the vertex of the corner angle.) Two separate votes are recorded if the shape may be on either the dark or the light side of the detected edge. The votes are added into an accumulator array consisting of one cell for each permissible shape instance.

Each cell of the accumulator corresponds to an (x, y) coordinate position in the image at which a shape center might be found. Each cell also has an associated rotation and radius, so that there may be many cells corresponding to each (x, y) position. The set of cells belonging to a single (rotation, radius) pair is called an accumulator plane. For efficiency the search may be limited to template center points spaced every few pixels in each image direction and to rotation and radius values that are also coarsely quantized.

Each edge votes for many potential instances of the target shape. Some cells in the accumulator will receive only a few votes while others will receive many. A cell

corresponding to a true instance of the shape would receive far more votes than other cells around it. Shape instances may thus be found as local maxima in the accumulator. The few maxima receiving the highest counts are reported as likely matches.

2.2. Typical Applications

The GHOUGH program may be used in any application where a known shape is to be found or measured in an image. It will detect instances of the target shape within a specified range of sizes or orientations and can do so in the presence of noise, occlusion, weak shadows, and some types of texture or camouflage. Matching is done in two dimensions, without regard for perspective changes or image warps. Large shapes are located more reliably than small ones, and high-contrast instances more reliably than those that match their backgrounds.

The program can detect or verify the presence of expected image features as in registration, terminal guidance, and cueing of specific targets. It is well suited for coarse registration (finding the first landmark) and for fine registration of individual landmarks, but not for simultaneously finding many different landmarks.

Once a target has been located, the program may be used for camera calibration or change detection. These applications require very fine quantization in the accumulator to accurately determine the rotation angle and scale of the landmark image. Camera position can then be derived if the landmark dimensions are known. Such information is useful in aerial image registration and in navigation.

Parameters measured from a calibrated imaging system may be used for image analysis and change detection. The GHOUGH program is only capable of finding matches to expected shapes. Analysis of shape changes, as in snow cover or lake boundaries, requires higher-level techniques. Better tools for analyzing close perspective views of three-dimensional objects may be found in the ACRON.M system [Binford81] and in various warp-registration systems [Chien75, Belsher79, Clark79, Lowe80, Noges80].

Temporal change detection and scene analysis problems often require that irregularly shaped regions of one image be identified as corresponding to regions in the other image despite contrast and location differences. This can be done by segmenting both images and establishing region correspondences [Price75], or by segmenting one image and using the GHOUGH program to find the same regions in the other image.

Tracking of large objects, as in weaponry fire control, is an application of image-to-image registration for which the GHOUGH program is well-suited. The shape can be extracted from the first image frame (or from a target database) and can be used to track the object through succeeding frames. Some augmentation of the GHOUGH algorithm is needed if perspective changes of three-dimensional objects are to be accommodated during tracking. It should be noted that tracking is a well-developed area of image processing and that many special-purpose algorithms already exist [Flachs76, WDESC76, Woolfson76, Choate79, Dougherty79, Fitts79, Flachs79, Goodman79, Maybeck79, Orton79, Pridgen79, Reid79].

The GHOUGH program is ideally suited for finding multiple instances of a shape, as in industrial bin picking, microscopic particle counting, or railroad yard monitoring. It might find use in counting the buildings or finding all the right-angled corners in a

high-resolution urban scene. Other straightforward applications are character recognition (a limited symbol set), industrial parts identification, and sensing the position of a robot arm. It could also be used to search for additional parts of an object or structure once a subpart has been found. Other edge-based template matching techniques [Hemami70, McKee76, Perkins76, Bolles82] may be competitive for all these uses.

A particular example of shape detection is the identification of circular structures in an image. Circles in aerial images often indicate storage tanks, while those in ground-based imagery may indicate vehicle wheels, storage tanks, or other man-made artifacts. The GHOUGH program can be used to identify all circles or circular segments in an image (within a specified range of radii and contrast parameters).

GHOUGH can also be used to find ellipses or parabolas of any orientation, although the aspect ratio of the shape must be fixed. Ellipse detection is used to find circular objects viewed from an oblique angle or cylindrical parts scanned by a light beam [Bolles81]. Hough parabola detection has been used for locating ribs in chest x-rays [Wechsler77]. More general techniques for finding parametric curves of unknown aspect ratio will be described later.

Texture analysis is the identification of a texture pattern or of the process that produced it. Some methods of texture analysis require that subpatterns (*i.e.*, texture elements, or "texels") be identified, as in the identification of an orchard by the regular spacing of the trees. The GHOUGH program is well suited to finding the texture primitives providing that they are all similar and have sharp outlines. The program cannot address the harder problem of knowing which texture elements to look for in the first place.

2.3. Potential Extensions

The GHOUGH algorithm is essentially a pattern recognition technique [Ballard81a]. Minor modifications to the data collection processes will change the class of patterns to be detected. For instance, a line detector could be substituted for the edge detector in order to locate thin rivers, roads, or other linear features in an image. One such approach that uses line segments instead of points or edges is described in [Davis82].

The current restriction that templates be specified as closed curves is not inherent in the algorithm. Future versions of GHOUGH may permit linear or arbitrarily connected templates. This would increase the reliability of detecting linear features and common shape fragments (*e.g.*, object corners). It would also permit internal detail to be used in a template as well as silhouette information.

A particular case of linear feature detection is the detection of all straight line segments in a scene. This is especially important in navigational obstacle avoidance, terminal guidance, and robotic manipulation. The Hough algorithms grew out of such applications [Hough62, Griffith70, O'Gorman73, Perkins73, Doudani77], and the GHOUGH program could easily be adapted to this purpose. Lines segments or wires found in a scene would have to be verified by other means since the Hough transform is not well suited to finding the end points of a segment.

Circle detection has been discussed above. Although the current program can find circles and other analytic shapes, it is not as proficient at these tasks as a special-purpose algorithm. If circle finding is important, it may be advantageous to use a

special accumulator for the purpose. This is very similar to the adaptation for line finding mentioned above. Hough curve detectors have been studied by several authors [Duda72, Merlin75, Kimme75, Ballard76, Tsuji77, Tsuji78, Sklansky78, Shapiro78a, Shapiro78b, Shapiro78c]. Least-squares fitting [Bolles81] and other pattern recognition methods have also been studied, and may offer advantages over Hough techniques. Once a curve or boundary has been located, Hough techniques offer one of many approaches to segmenting the curve into meaningful fragments [Shapiro79b].

Tracking point targets is easier than area-based change detection. Hough techniques can be used to cluster motion or optic flow vectors to find groups of objects moving in formation [O'Rourke81b]. Other generalized Hough techniques have been used to track changes in line drawings or edge maps and to check multitarget trajectories for consistency [Yam81].

Hough methods can also be used for finding vanishing points in perspective scenes and foci of expansion in optic flow images. A method has been developed [Ballard81b] that uses a Hough line finder on the original image and a similar circle finder on the accumulator. This works because all lines through a vanishing point in the original image will cluster along a circle through the origin in Hough space. It may be possible to develop a single-stage Hough-style transform to replace this two-stage process.

Several Hough techniques are candidates for detecting complex composite shapes. Articulated shapes, such as the arm of a robot manipulator, are best found by locating the individual parts and then using higher-level knowledge to recognize the composite. Complex rigid shapes, such as a vehicle, may be found by a similar technique or by convolution in the accumulator [Ballard81a]. The convolution technique is based on the fact that an edge element votes more accurately for nearby reference centers than for template centers further away. One can therefore increase recognition accuracy by breaking the template into smaller chunks, voting for the chunks independently, and then combining the votes according to the template decomposition. This permits recognition in cases where individual subparts may not be identifiable without contextual information. Another approach, called the hierarchical generalized Hough transform, has been developed by Davis and Yam [Davis82].

Several researchers [Ballard81b, Ballard81c, Hinton81, Sabbah81] are now working on generalizations and implementations of the Hough transform for intrinsic image estimation. This is the conversion of an image, or a sequence of images, into explicit maps of edge locations, color regions, surface orientations, distances from the viewer, optic flow, etc. It is believed that all of these intrinsic image properties must be derived simultaneously (*i.e.*, the problems are tightly coupled) and that only massively parallel architectures such as the human brain are adequate for the task. These intrinsic images have been labeled "parameter networks" [Ballard81b], and the associated "constraint transforms" are modeled as Hough transforms. New methods of image analysis may arise from this work.

2.4. Related Applications

This section describes applications that are similar to Hough transform applications, but that differ in some fundamental fashion. While the difficulties with applying the transform could be overcome with special hardware, other techniques would often be more appropriate.

Cueing is the initial detection of interesting objects in a scene. Sometimes object

recognition is included with cueing when only certain types of object are of interest. The GHOUGH program can be used to cue instances of a given shape, but it is not well adapted to recognizing classes of objects. At best it can maintain a separate accumulator for each of several shapes or subshapes (*e.g.*, corners, straight lines, or parallel segments). Cueing is better performed with real-time level slicing or pseudo-coloring, or with matched filters [Otazo79], statistical classifiers [Touchberry77], blob detectors [Klein77, Tisdale77, Milgram78a, Deal79, Tisdale79], corner detectors [Perkins73, Rosenfeld78, Hwang79], or unusual-pattern detectors [Haralick75, Winkler78].

Object identification is the recognition of a shape once it has been cued. The GHOUGH program is only capable of template matching [Stockman77], and would have to maintain a separate accumulator for each identifiable silhouette. In gray-scale imagery this may be less successful than correlation matching; in perspective scenes the more flexible techniques of boundary following [Lucey76, Martelli76, Nahi78] or edge linking [Nevatia76, Milgram78b, Narendra79, MacVicar-Whelan81, McQueen81] and shape analysis [Arcelli71, Freeman74, McKee76, Tanimoto77, Binford81] may be superior.

Image-to-image registration is the process of finding the warp coefficients needed to align one image with another. A GHOUGH approach would require that (possibly unidentified) shapes found in one image be located in the other image so that warp parameters could be computed. This fails on low-contrast or blurred imagery and is less robust than correlation methods using all the gray-scale or gradient information in the scene.

Stereo compilation is a particular type of image-to-image registration in which the two images are known to differ by a small change in camera position. The compilation uses the known perspective change to compute distances from the camera to elements in the image. This and other stereo matching problems are basically one-dimensional, and are better handled by correlation or special-purpose techniques [Gennery77, Baker80, Grimson80] than by a general shape-detection routine.

Registration and cueing are just particular applications of pattern recognition. A great many pattern recognition techniques have been developed for recognizing regularities or particular patterns in lists of data vectors. Any of these techniques could be substituted for the accumulator technique used by the GHOUGH program. Some of these techniques offer adaptive improvement as they process new images, others are guaranteed optimal for particular detection problems.

Ellipse detection can be used as a particular example. The GHOUGH program treats ellipse templates as general shapes, no different from any others. Each edge element must vote for many cells in the accumulator -- all the cells occupying a cone in (x,y,radius) space. These votes could be reduced to a single vote if edge pairs were considered instead of individual edges [Laws77]. Computational savings are even greater if ellipses of arbitrary aspect ratio are to be detected in a five-dimensional parametric space [Tsuji77, Tsuji78]. The tradeoff in these cases is that the number of edge pairs is much larger than the number of edge elements, so that complex screening is needed to reduce the number of edge pairs actually used.

Another advantage of some pattern recognition techniques is that the detected object is identified by arbitrarily accurate coordinates. There is no quantization problem or corresponding computational space/time tradeoff as with the Hough techniques. Two-pass coarse/fine registration techniques are thus not required, although back projection and validation against the image may still be desirable.

Section 3

Description

This section presents the history of generalized Hough analysis methods and a detailed statement of the GHOUGH algorithm. The historical information is intended to clarify the major issues in recursive segmentation and to provide entry points into the literature.

3.1. History

The GHOUGH program detects instances of a given shape by combining many pieces of local evidence. The original Hough transform [Hough62, Rosenfeld69, Griffith70] was a method of finding lines through a pattern of points. Duda and Hart [Duda72] introduced the normal parameterization of lines instead of the slope-intercept form. Others [O'Gorman73, Perkins73, Shapiro74] used gradient direction at each point to improve the voting method.

Other related methods have been used to locate circles [Bastian71, Duda72, Kimme75], parabolas [Wechsler75], and ellipses [Tsuji77, Tsuji78]. The extension to other analytic shapes was obvious, although implementation questions remained [Shapiro75, Shapiro76, Shapiro78a, Shapiro78b, Shapiro79a, Zabele79]. A survey of this work has been published [Iannino78]. Hough transforms have also been applied to trajectories generated by dynamic processes [Shapiro78c, Yam81].

Merlin and Farber [Merlin75] extended the Hough technique to nonanalytic shapes by using a template to control the gathering of evidence. Their method mapped each (directionless) point in a binary image into an inverted trace of the template in the accumulator. The best instance of the sought shape was then indicated by the cell accumulating the most votes.

Ballard [Ballard81a] restructured this technique to make it flexible and efficient. The GHOUGH program grew out of this work [Sloan80]. It uses a compiled template to simplify addressing in the accumulator, and it starts with directed edge segments instead of image points. The directed edge segments constrain the possible orientations of a shape instance so that only a few cells need to be incremented.

3.2. Theory

There have been many formalizations of Hough transform theory, particularly for the application of analytic curve detection. See van Veen [van Veen81] for a particularly nice analysis of straight line detection; a related error analysis has been published by [Shapiro79a]. It has recently been recognized that Hough transforms are a special case of the Radon transform used in computerized tomography [Deans81]; this may provide important theoretical methods similar to those of Fourier analysis. Fourier theory may also be applicable since the Hough voting pattern is similar to an optical point-spread function [Brown82a].

Description

All Hough techniques consist of the following elements:

- A shape specification with N-dimensional parameter space P;
- An N-dimensional accumulator, A, representing P;
- A local element detector, E, applied to the image;
- A voting rule, V, mapping the information of E into A;
- A detection rule, D, specifying conditions on A that indicate instances of the shape.

The GHOUGH program makes specific choices for these elements that determine its range of application.

Two image regions have the same shape if the outline of one can be generated by a combination of translation, rotation, or uniform scaling of the other. Each such set of equivalent shapes is represented by a template of normalized size and arbitrary orientation. (The template is scaled to have unit radius relative to the reference point described below.) A particular shape instance is thus represented by a particular template and four parameters: horizontal and vertical position, rotation, and radius.

A template differs from a shape in that it has an associated reference point. The reference point is selected by the user when the template is first digitized. (It could be selected automatically, *e.g.*, as the center of boundary mass.) Usually the reference is an arbitrary point near the center of the shape, but it can correspond to any invariant location or feature. Although it need not be inside, computational problems will be minimized if all distances to the boundary are small.

A template is also composed of disconnected points instead of a continuous outline. The points are usually entered by tracing an image region, possibly with the aid of spline interpolation. They can also be generated mathematically since they are completely independent of any gray-level information in the image. The points must trace the shape clockwise, and the sequence is currently required to closed on itself. The density of sampling along the shape outline is arbitrary, but the smoothness of the representation may be critical. Boundary direction through a point is assumed to be parallel to the line through the point's two nearest neighbors. A small error in this direction can result in poor search performance because of magnification in extrapolating to the template's reference position.

The generalized Hough technique compiles the template into a data structure called an R-table. This is a sparse representation of the object-centered location and orientation of each template edge point, stored as a list of lists. The main list is accessed by the rotational orientation of an edge element relative to the horizontal axis. Associated with this orientation is another list of vectors representing x and y offsets from the edge element to the template reference point. Duplicate or multiple offset vectors may be present. Together all the edge orientations and offset vectors are sufficient to reconstruct the template.

In operation, an edge detector is applied to all pixel neighborhoods in an image window. At present a modified Sobel operator is used. A Sobel operator measures the gradient at each image point by the root-mean-square response to two convolution masks:

```
-1  0  1        -1 -2 -1
-2  0  2         0  0  0
-1  0  1         1  2  1
```

Description

The modified Sobel uses the sum-of-absolute-value combination of the two weighted sums. Experience has shown that the two methods are nearly equivalent, although the root-mean-square method is a little more constant in its response to rotated edges.

The response at each pixel is compared to a threshold supplied by the user; if it is above threshold, the detected edge is permitted to vote for likely shape centers. The proper threshold setting depends on the scene content, the target to be detected, and the edge detector itself. The modified Sobel detector typically produces a log normal distribution of edge gradient strengths.

The edge direction is computed from the arctangent of the responses to the two Sobel masks. A detected edge is then (conceptually) rotated by the minimum permitted rotation for valid shape instances. This orientation gives an index into the R-table to find the associated list of offset vectors. The accumulator cells corresponding to each offset vector and permitted radius (scale factor) are incremented. (A separate accumulator plane is used for each rotation and radius.) Then the edge orientation is further rotated by the specified increment and the indexing and voting procedure is repeated. This continues until the full range of permissible shape rotations has been processed.

Thus any given edge-element may vote for many possible shape centers. The number of accumulator cells incremented is proportional to the number of scale factors and rotational angles permitted in the search. The accumulator cell with the most votes indicates the most likely location, rotation, and radius of the shape.

Since there may be any number of shape instances present, including none, it is necessary to decide which cells correspond to genuine occurrences. The GHOUGH program uses two criteria: each reported shape instance must be a relative maximum within the four-dimensional accumulator, and must also be above some threshold number of votes. A relative maximum is simply a cell with at least as many votes as neighboring cells. The threshold is required to prevent isolated votes (*e.g.*, single votes surrounded by no others) from being reported. At present all cells meeting these criteria are reported as shape instances, with those having the most votes reported first. Any higher level analysis is left to the program receiving this list of "possible target hits."

Section 4

Implementation

This section documents the SRI Testbed implementation of GHOUGH. It is intended as a guide for system maintainers and for programmers making modifications to the GHOUGH system. The terms used in this section may be a little cryptic: they are either defined elsewhere in this report or come from the supporting operating systems.

The SRI Testbed uses the EUNICE operating system, which is a Berkeley UNIX[1] emulator for VAX computers using DEC's VMS operating system. EUNICE was developed at SRI to permit simultaneous access to UNIX and VMS software and system services, and to implement improvements to UNIX such as significantly faster image I/O. EUNICE is now a commercial product maintained by The Woolongong Group in Mountain View, California.

The GHOUGH package is currently compiled as a single file, *ghough.c*, containing a main program and several subroutines. The underlying algorithm is very little changed from the original University of Rochester implementation, but the command interpreter, help system, and supporting image access and display utilities are all new. These external routines are all part of the Image Understanding Testbed environment.

The main program and related files are in directory /iu/tb/src/ghough. Subdirectories *help* and *srchhelp* contain the text files used by the help system at the top command level and during the search command. Directory *template* contains predefined template files, and *demo* contains some simple shell scripts and "ground truth" files used in the evaluation effort.

Source code and help files for the CI driver are in /iu/tb/lib/cilib. For extensive documentation type "man ci" or run "vtroff -man /iu/tb/man/man3/ci.3c". The CI driver uses command-line parsing routines in cilib/cmuarglib and in /iu/tb/lib/sublib/asklib; both of these may someday be replaced by the Testbed argument parsing routines in sublib/arglib.

Other utility routines contributed by CMU have been distributed to /iu/tb/lib/dsplib/gmrlib, /iu/tb/lib/imglib, and /iu/tb/lib/sublib, and are documented in /iu/tb/man/man3. Some of these have been modified or rewritten for the Testbed environment: the image access code, for instance, reads Testbed image headers as well as CMU image headers.

To compile the GHOUGH program, just connect to this directory and type "make". You may type "make -n" to see what will happen if you do this. Additional options are documented in the header of the makefile.

The program contains two edge-detection routines. The first is a modified 3x3 Sobel operator; the second is a generalized 3Nx3N Sobel operator, where NxN block averages are substituted for the single pixel values used in the 3x3 Sobel. Both routines are

[1]UNIX is a trademark of Bell Laboratories.

meant to be applied at arbitrary image positions, as opposed to being optimized for moving-window application to an entire image plane. The interactive driver allows either edge operator to be applied to any image point. (This may help in threshold selection.)

The original contribution used a 6x6 version of the Sobel edge detector in which each 2x2 subsquare was averaged to produce the 3x3 pattern used in an ordinary Sobel edge detector. This gave noise immunity and increased accuracy for large targets at the cost of much slower operation and of poor performance on small targets. The Testbed version has been recompiled to use the standard 3x3 operator.

The search algorithm itself is separated from the user interface routine that gathers the search arguments and checks them for errors. The search algorithm could thus be invoked by any other program that set up the proper environment. It takes an image and several arguments as input and returns an accumulator as output.

The accumulator analysis has also been separated into independent modules. First the accumulator is searched for local maxima. The list of maxima is sorted and then passed to a deeper level of the CI driver. The user can then apply any sequence of analysis and display commands to the sorted list.

The original contribution reported only the single shape instance receiving the most votes. SRI has modified the testbed version to report a list of matches ordered by the number of votes. This permits identification of multiple instances in one image, as when finding oil storage tanks in aerial imagery. It was also necessary to write routines for examining and saving the list of matches, and for comparing the list to a ground-truth database.

GHOUGH demonstrations have been set up in subdirectories *lake*, *circle*, and *ellipse* of /iu/testbed/demo. Just connect to the appropriate directory and run the *demo* command.

The *demo* program in the *lake* directory is fully automatic: just type 'demo' and it will proceed to find the lake. You may also run the *display* script to show the results of a previously run *demo* script. For more interaction, invoke the GHOUGH program directly (*e.g.*, by typing 'ghough') and issue the '< lake.cmd' command; you will then be asked to delimit a search area with the cursor and to provide other parameters and display commands.

The *circle* and *ellipse* directories are very similar, except that there is no predefined interactive version of the demonstrations. The ellipse finder takes a particularly long time to run, so you should normally just run the *display* sequence.

Section 5

Program Documentation

This section constitutes a user's guide to the GHOUGH package as it is implemented on the SRI Image Understanding Testbed. As with any reference manual, it has occasionally been necessary to refer to terms before they are defined and discussed in detail. A preliminary scan through the section may be helpful on the initial reading. Additional information is available on-line, as described below.

5.1. Interactive Usage

The GHOUGH algorithm is currently embedded in a subroutine to fill the accumulator with votes and another subroutine to analyze the vote pattern and report back likely shape instances. These routines are called by ghough.c, a driver package that allocates the accumulator array, sets up the required environment, and interacts with the user. The following is a sample session using the GHOUGH program.

```
ghough
```

This invokes the program. You need not specify the full directory path name for the executable file if the path is given in your standard ".cshrc" shell startup file. If you have no startup file, you may have to specify "/iu/tb/bin/ghough" or some other full path name.

```
Generalized Hough Transform
>
```

The system responds and waits for commands. GHOUGH uses a version of the CI driver, so that built-in features of that driver are available. Additional commands are provided for the GHOUGH image analysis capabilities.

```
> *

  draw                search
  erase               select
  open                sobel
  print               template
  quit                trace
                      help


> *=

  verbose         no
```

An '*' command lists all of the GHOUGH commands, and an '*=' command will list any user-accessible variables. Typing 'help' will give further information on the CI

command interpreter and the help subsystem, and 'help *' will give a list of all available help topics. Additional information may be obtained by typing help and the name of a topic. For example:

```
> help menu

draw ---------- draw the template.
erase --------- erase the overlays.
open ---------- display the input picture.
print --------- print the template table.
quit ---------- clear the display and quit.
search -------- find instances of the template shape.
select -------- select one of the four overlay colors.
sobel --------- find Sobel gradient at specified point.
template ------ read a template file.
trace --------- trace a shape to create a template.

Use "verbose = yes" to enable additional printout
during searches.


> help usage

Commands issued in the following order should work.

  open ---------- display an image.
  template ------ read a template file.
  draw ---------- display the template shape.
  search -------- search for instances of the shape.
  quit ---------- clear the screen and terminate.
```

To open an image file and display it, type

```
> open

Image file:  <bw.img>
```

The default name is bw.img, but any other name may be specified. The image name may also be given on the command line, as in "open /iu/tb/pic/plant/bw.img". The image may then be used for tracing a new template or for searching using an existing template.

To create a new template:

```
> trace

Draw a closed shape (clockwise).
Use <CR> to enter a point, q to quit.

Specify the shape center, then <CR>.

  Reference point: [256, 256]
```

```
Enter points with <CR>, terminate with q:

  Point  0: [156, 256]
  ...

Number of points entered: 46.
Maximum radius = 63.49.

Output file name: <template.dat>
```

The x and y components of the distance between each edge point and the arbitrary center of the shape are stored in a template file. For help in using the cursor, type 'help cursor.'

Any number of points may be used to outline the shape -- the more points that define a shape, the more are available to vote for the shape. A large number of points in the R-table also causes GHOUGH to run slower. One is faced with a trade-off between computation time and precision in detection of the desired shape.

```
> template

  File name:  <template.dat>
```

An R-table is built from a stored template file. The default file name is template.dat, but you may specify any valid template file. This command is not needed if you have just entered a template by tracing an image object.

To draw a shape instance in the display overlay:

```
>  draw

  Center column:   (0 to 511) <384>
  Center row:      (0 to 511) <128>
  Template angle (degrees):   (-359 to 360) <0>
  Template radius (pixels):   (1 to 511)  <60>
```

The default position is in the lower-right quadrant of the screen, which is convenient if the image you have open is 256x256 or smaller. The template angle is a counter-clockwise rotation relative to the original orientation of the traced shape. Template radius is the radius of the largest circle centered on the reference point that just contains the shape.

To run the edge detector over the image and search for shape instances:

```
>  search (160,90),(210,170)
```

This command begins a search for the template shape in a window with the specified lower left and upper right corners. If the corner coordinates are not specified, they will be requested interactively, you may then use the cursor to indicate the processing window. The remaining arguments of the search command are given in interactive form below.

The search command builds an accumulator and fills it with votes for template instances. The accumulator may be regarded as a stack of two-dimensional arrays, one for each combination of acceptable template size and rotation. The stack of arrays is registered with the image window specified above. Shapes to be found need

not be entirely within this window, but the window must cover the shape centers.

```
Horizontal resolution:   (1 to 51) <1>
Vertical resolution:     (1 to 81) <1>
```

You are next asked for the spatial resolution of the search. Specify single-pixel resolution [the default] for the finest resolution currently available. Use a larger spacing in either direction for a coarse search that uses a smaller accumulator and requires less search time.

```
Minimum rotation (degrees):   (-359 to 360) <-3>
Maximum rotation (degrees):   (-3 to 360) <3>
Angular Resolution (degrees):   (1 to 7) <1>
```

Specify the acceptable range of template rotations and the angular resolution of the search. You may specify any whole number of degrees. Here we have allowed three degrees on either side of the original template orientation.

```
Minimum radius (pixels):   (1 to 999)   <60>
Maximum radius (pixels):   (60 to 999)   <66>
Radius resolution (pixels):   (1 to 7)   <1>
```

You must also specify the acceptable range of radii for instances of the template shape. This is a distance in pixels and not a magnification factor. The radius is that of the largest circle which just contains the shape. The program is expecting floating-point numbers and will quantize the template radius to any specified resolution.

```
Edge threshold:   (0 to 999) <120>
```

You must specify the weakest edge gradient that will be allowed to vote for potential template instances. You may specify 0 if you want to use the best edge fit at every image pixel, but this will slow the search and raise the noise level in the accumulator. Fewer edges are considered as the value of the threshold increases. You may want to use the Sobel command (before invoking search) to establish a practical value for the threshold.

```
Contrast (b/w/m):   <black>
```

Specify the contrast of the figure as black, white, or mixed. This argument determines how the edge gradient will be interpreted. If the figure appears dark on a light background then you should specify black; if it is light on a dark background then specify white. You may specify mixed to include either or neither of the conditions. each edge element will then vote for both a dark shape and a light one.

```
Examining the edge structure ...
```

The program is now searching through the window looking for edges above the specified strength threshold. Each edge that it finds is displayed on the monitor screen.

```
Searching for local maxima ...
```

The program is now searching the accumulator for local maxima. A list of these potential template instances will be made available for further analysis.

```
37 maxima found; highest count is 5.

Enter display commands:

>
```

The program has found 37 potential hits, and has created a new subcommand interpreter so that you may study the results before terminating the search routine. To see the available subcommands:

```
> help menu

  above --------- display matches above a given count.
  display ------- display a particular match.
  erase --------- erase the overlays.
  first --------- display the first n matches.
  level --------- display a particular level of matches.
  quit ---------- clear the display and quit.
  save ---------- save the overlay as an image file.
  score --------- tabulate "hits" in a ground-truth file.
  select -------- select one of the four overlay colors.
  top ----------- display the first n levels of matches.

  Use "symbol = yes" to turn on full-shape display, "no"
  to display only center points (except for the "display"
  command).
```

This 'help menu' command shows the names of commands available at this level. Enter '*' or '*=' for a more concise list, or 'help topic' for more information.

```
> save edges.ovl
```

If you want to save the edge overlay you should do it now. The current program cannot recreate this display if you go on to something else.

```
> first 4

 #    Column     Row     Angle    Radius   Count
 1      181      141       2       66.0       5
 2      181      143       0       61.0       4
 3      181      144      -2       64.0       4
 4      181      143       0       62.0       4
```

Here we have listed the first 4 local maxima. The count field shows the number of votes that each match received. (Note that the fourth is not necessarily better than the fifth.) The centers of these four matches are displayed on the screen.

```
> top 2
```

#	Column	Row	Angle	Radius	Count
1	181	141	2	66.0	5
2	181	143	0	61.0	4
3	181	144	-2	64.0	4
4	181	143	0	62.0	4
5	181	139	0	64.0	4
6	181	144	-1	64.0	4

The top command displays a number of threshold levels in the accumulator. Here the top two levels are displayed. We could save this display with a save command.

```
> level 1
```

#	Column	Row	Angle	Radius	Count
1	181	141	2	66.0	5

We can also display a single threshold level. 'Top 1' and 'level 1' give the same result, but the two commands differ for larger arguments.

```
> display 1
```

#	Column	Row	Angle	Radius	Count
1	181	141	2	66.0	5

Here we have displayed a particular match (the first). Both the shape center and the outline are displayed on the monitor screen.

You may also use the 'score' command to compare detected shape instances with a previously existing ground truth file. This file is currently just a list of all known instances of a particular target type giving position, rotation, and radius; for the lake in the plant image, this is just

```
181  143  2  64.2
```

The score command will search for this target among the first 40 detected shape instances. These commands are particularly useful for later evaluation of GHOUGH runs executed as background or batch jobs. For our sample session:

```
>      score lake.tru

 #    Column      Row      Angle     Radius    Count

        181        143        2       64.2    (known)
 1      181        141        2       66.0       5
 2      181        143        0       61.0       4
 3      181        144       -2       64.0       4
 4      181        143        0       62.0       4
 5      181        139        0       64.0       4
 6      181        144       -1       64.0       4
 7      182        142       -1       60.0       4
 8      182        142        0       60.0       4
 9      181        143        0       60.0       4

..      ...        ...       ..       ....       .

1 of the 1 known targets were found
in the best 37 Hough maxima.
```

This example shows the lake being reported as many matches, each differing only slightly in its combination of parameters. The GHOUGH program recognizes these matches as being equivalent, but does not average or otherwise combine them to get a more accurate determination.

The quit return terminates this driver level and gives control back to the search routine. The search routine offers one more service before returning control to the top-level command driver.

```
> quit

Output a Hough plane?  <no>
```

A plane of the accumulator array may be output as an image file for later display. This can be useful for debugging or for getting a feel for the nature of generalized Hough analysis.

```
> quit

Finished.
```

A quit command terminates the session. The monitor screen is cleared and the display processor is freed for other users.

5.2. Batch Execution

The GHOUGH program offers two methods of invoking prestored commands. The first is the invocation of CI command files. For example, you might give the command

```
> <lake.cmd
```

where the file lake.cmd contains the commands

```
open /iu/tb/pic/plant/bw.img
template /iu/rochester/template/lake2.dat
draw 384,128,-45,45.0
search .,1,1,-3,3,1,60,66,1,120,black,no
```

In this case the GHOUGH program will open an image, read a predefined template, display the template, and search for matching instances. You will be asked for the search area since its coordinates were omitted from the search command. You will also be able to enter interactive commands to examine the search results and save the corresponding graphic displays.

The second method is to drive the entire GHOUGH session from an operating system script. A UNIX C-shell script might look like:

```
# Run the lake demo.
ghough <<!
  open /iu/tb/pic/plant/bw.img
  template /iu/rochester/template/lake2.dat
  draw 384,128,-45,45.0
  verbose = yes
  search (160,90),(210,170),1,1,-3,3,1,60,66,1,120,black,no
    save edge.ovl
    symbol = yes
    first 20
    top 1
    save top1.ovl
    top 4
    save top4.ovl
    display 1
    save match1.ovl
    score lake.tru
    quit
  quit
!
```

This script is designed to run without user interaction. It does display its output on the terminal and on the display screen, but does not wait for you to look at the data. (You may add 'push_level' commands wherever you want it to pause and wait for commands. A quit command will then return you to the script command file. Specifying 'quit 2' will terminate the entire run.)

The script example above contains save commands to save copies of selected graphic overlays for later viewing. To save the typed terminal output you should pipe the standard output to a file. The UNIX method for doing this is to add " >session.log" to the ghough command within the script or to the UNIX command line that invokes the script. You may also use the UNIX 'script' or 'tee' commands to route the typed output to a file and to your terminal.

The actual submission of this shell script is described in the UNIX Programmer's Manual. You should run it in foreground mode if you want to interact with the program. If you run it in background mode, be sure to pipe the output to a log file so that it won't appear on your terminal. On a UNIX system you can monitor the log file during execution (using a 'cat' or 'tail -f' command) to make sure everything is running smoothly, and you can halt the process or reconnect it to your terminal if you wish. At present there is no way to turn off the GHOUGH display commands, although this option is planned.

Section 6

Evaluation

This section documents the performance of the GHOUGH program in several hundred test runs on a variety of aerial imagery. We have extracted rules for using the current system and have evaluated its characteristics.

6.1. Parameter Settings

GHOUGH search options available to the user are presented below. Typical parameter values are specified and the effects of different values are explained.

Image Content:

The GHOUGH program is at its best finding large, well-defined silhouettes in otherwise unvarying imagery. Since this is not a very useful talent, it is well that the program is also able to find partially hidden objects in noisy or blurred images. Any type of imagery is suitable as long as a sufficient portion of each target boundary is detectable. The objects must usually be obvious to a human before GHOUGH will be able to find them reliably.

The quality of object detection can be partially controlled by setting a threshold (described below), but is ultimately limited by the edge detector used. The modified Sobel detector currently in use works very well on most imagery, but is not as sensitive as an adaptive or task-specific detector might be. It does not give subpixel resolution, and the gradient angle is not very reliable (especially for weak edges).

A particular requirement with this edge detector is that object edges be sharply defined. Objects with gradual, uncertain boundaries (*e.g.*, reservoir outlines or FLIR targets) or strong internal gradients (*e.g.*, domes or "hot spots") will usually be located, but the position and orientation reported may not match that chosen by a human. The reported shape may also depend on the edge threshold used.

The requirement of sharp edges does not imply that smooth, continuous object boundaries are required. The program is quite tolerant of noise in the outline and is able to find irregular, incomplete, or discontinuous shapes. The circle template, for instance, often responds to forest clearings, tree tops, road intersections, and curved embankments, as well as to square buildings and to image "hot spots." The irregularities in these image structures spread the vote cluster in the accumulator, but the local maximum may still be above the general noise level.

Shadow edges usually fit the requirement for strong, sharp edges. It is often easier to find a shadow than to find the object that cast it. This may be a useful cueing technique, but must be used carefully to avoid reporting objects at incorrect locations. A similar problem exists with high-resolution imagery: the position reported for a part of an object (*e.g.*, the circular top of a storage tank) may not correspond

to the position of the whole object.

These characteristics mean that the program is best suited for three tasks: locating industrial parts in high-contrast imagery; counting numerous, obvious, similar objects such as storage tanks, barracks, or microscopic particles; and precisely positioning a template when an approximate location is cued by the user or by another system. Even for these applications, the program must be supervised and its output edited. Other applications will require further development of the technique.

Template:

Template shape is dictated by the targets being sought. Although this is not under user control, it may be helpful to understand the effect of shape complexity on GHOUGH performance.

A template is first compiled into a R-table, and all matching is done against this data structure. The R-table is sorted by edge angle, with one or possibly more shape center offsets recorded for each angle. For a convex template these offsets form an orderly progression matching the order in which the points were traced. For an involuted template (*e.g.*, a star-shaped outline) they jump around as points from one part of the trace interleave with points from another part. The net result is that, during edge processing, the votes from a particular image edge also jump around as different rotations are considered. This increases paging activity and total execution time, but has very little effect on detection performance.

The performance of a template depends only weakly on the number of points it contains. The density of points along the template perimeter determines the accuracy with which template edge angles are known. This is because the angle at a particular template point is computed as the direction between its two neighbors. The density of points also determines the accuracy of accumulator votes since the template point used by an image edge element for its voting is the one with computed angle nearest to, but larger than, that of the image edge.

These two effects cause spreading of the accumulator votes for a given shape instance. For small targets (*e.g.*, seven pixels in radius) the spread is slight, but possibly significant because the number of votes is small. For large targets (*e.g.*, 64 pixels in radius) the spread is much greater but is balanced by the large number of voting edges. A template with as few as eight points can be used to find targets, but will find a large target only if one of the accumulator planes represents exactly the right rotation and radius and the density of detected edges is almost optimal. For more robust operation the number of template points should be at least half the target radius. Generally 32 points are quite sufficient.

Contrary to intuition, using more points in the template neither increases the execution time significantly nor improves performance. The only effect on execution time is that of indexing into the R-table to find the closest template edge: this is currently done by following a chain of pointers until the next larger angle is found, but the program could be written using a hash table or array structure for the R-table that would have a constant (and faster) accessing time.

The effect on performance is that more accurate template edge angles slightly improve vote clustering and rotational accuracy. Improved vote clustering is only important in borderline cases and might be better achieved by convolution smoothing of the accumulator prior to (or during) detection of the local maxima. Rotational

accuracy is more a function of target shape than of the accuracy of individual edge angles. Further, too dense a template will suffer from quantization effects as described below.

The most common method of creating a template is to trace a known target. The GHOUGH program has a cursor driver to make this possible. Unfortunately, the position of the cursor can only be read to the nearest pixel. For a sparse outline of a large target this makes no difference, but for a dense outline it means that a template point will usually differ from each of its neighbors by only a single vertical or horizontal pixel (or both). Thus the edge angle computed using its neighbors will usually be a multiple of 45 degrees. This introduces severe quantization errors that make it difficult to extrapolate the shape center from a given image edge element.

Another consequence of this quantization error is that the R-table will contain only a few angles, each having a list of many corresponding shape center offsets. This is in contrast to a normal template having many angular entries with only a few having multiple shape center offsets. The quantized template results in far more votes being cast, but the votes are very scattered and serve mainly to increase the accumulator noise level. (One test with a 256-point template produced cells with noise counts of 37 votes, compared with only 6 votes for a similar 32-point template.)

Four solutions to this quantization problem are obvious:

- Use fewer template points.
- Expand (zoom) the prototype before tracing it.
- Smooth the entered points using spline functions.
- Use each entered point as an approximation, and compute the true edge position and angle from the underlying image.

All of these methods could be used together. Only the first is currently supported by the GHOUGH software.

The quantization effect also occurs naturally for angular template shapes. A square, for instance, has four principal edge directions plus four minor directions at the corners. Each of the major directions is associated with numerous center offsets, and smoothing methods will not change this.

The quantization effect for an angular template increases the accumulator noise level. (The number of votes for a 10-point by 10-point square template is about nine times the number for a similar, smoothly curved one.) It also produces some surprising rotational effects due to angular rounding when indexing into the R-table. Suppose that a square template is used to find a square target against a uniform background. If both are aligned, the edges from the image will vote for points along the sides of the template square. If the image is tilted, say, 40 degrees to one side, the edges will vote for exactly the same template points and there will be little harm. If the image is tilted one degree in the other direction, however, the image edges will all vote for the corner points in the template. This will completely change the pattern of votes in the accumulator. The algorithm is robust enough that it may still report the correct position for the square, but the estimate of the rotation angle is likely to be incorrect.

This effect is noticeable when using a "corner" template to locate right angles in an urban scene. In one case the search orientation had been set for zero to 270 degrees in 90-degree increments. GHOUGH found a great many right angles, but reported some of them at 90 degrees off the true orientation. Further, any nearly vertical or

horizontal building edge could produce a spurious corner if some of the edge element directions were rounded up 90 degrees from others. This rounding effect will have to be corrected before GHOUGH can be used as a reliable corner finder.

Search window:

The user is asked to define a search window by specifying or pointing to a lower-left pixel and an upper-right pixel. The box connecting these defines an edge-detection area. The number of possible edge locations is thus

$$Window\ points = (columns)(rows)$$

The current 3x3 edge detector requires an additional one-pixel border of data; for this reason the edge-search window cannot be the entire image.

The user must also supply horizontal and vertical "search grains." These affect only the accumulator, not the edge-detection process. A search grain of 2 in each direction means that a single accumulator cell will be used for each nonoverlapping group of four pixels at unit search grain. This reduces the accumulator storage by a factor of four, and also reduces execution time, but limits the accuracy with which a shape instance may be found.

The number of positions at which a shape could be reported is

$$Template\ positions = \left(\frac{window\ size}{search\ grain} - 2\right)^2$$

for a square search window with side length divisible by the search grain. The 2 subtracted from the quantized window size represents a one-cell border around each accumulator plane. This border is used for indexing efficiency during searches through the accumulator. Future versions of the program may eliminate it or add it on instead of subtracting it from the specified window.

The general rule for setting the window size is to use the smallest possible window. It is usually faster and more accurate to do several small searches rather than one large one that includes targetless areas. A preliminary coarse search could be used to cue areas for more accurate searches.

A coarse search gives less positional accuracy than a fine one, but does not necessarily give poorer detection. Small errors in estimating edge directions (in either the template or the image) can produce large errors in estimating the shape center position. Often the smoothing due to spatial quantization overcomes the resulting cluster spreading. The search grain should be set to the largest value that will give acceptable resolution: 10% of the minimum target radius is suggested.

Orientations:

The user is also asked to specify the acceptable ranges for rotation and radius. Each is given as a minimum, maximum, and increment, where the increment behaves very much like the spatial search grain specified above. Again, specify the coarsest search that will give acceptable information. If targets differ greatly in size, it is advisable to do one search for small targets using a small radius increment and another for large targets using a large increment.

Evaluation

The total number of orientations permitted is

Orientations = (*rotations*)(*radii*)

Each orientation requires an accumulator plane with the number of template positions given above; thus

Search volume = (*orientations*)(*template positions*)

An additional one-cell border has been added to the search volume (rather than subtracted from it), so the total memory required is

Accumulator planes = (*rotations* +2)(*radii* +2)

Accumulator volume = (*window points*)(*accumulator planes*)

Detection performance in each plane is nearly independent of that in adjacent planes unless the rotation or radius increments are very small. Adding additional range or resolution in x, y, rotation, or radius does not degrade performance in the other dimensions. It does result in more potential "hits" being reported for the same number of targets because the Hough voting method often produces four-dimensional ridges or plateaus of identical counts. Any such cells that are not adjacent to higher cell counts will be reported as local maxima. Thus a finely quantized accumulator will often produce a pattern of false matches with lower counts around the true match cell. A fairly sophisticated four-dimensional analysis technique would be needed to suppress these false matches.

Template radius is specified as a floating-point number and may take fractional values. All other parameters are currently restricted to integer values. Requesting subpixel accuracy in the radius will occasionally improve detection performance, but the true target size will still be estimated poorly because target center position cannot be known to subpixel accuracy. The chief result of such a request is to increase accumulator size and execution time.

Threshold:

The GHOUGH program uses a modified Sobel gradient operator for edge detection: the root mean square of the horizontal and vertical gradients has been replaced with a sum of absolute values to make the operator faster. Gradients, or edge strengths, computed with this operator have an approximate log normal distribution with an image-dependent peak somewhere from zero to 1800. An image of cloudy sky may have a peak near ten; modal edge strength in a low-angle urban scene may be 200 (depending on contrast ratio).

The edge threshold should ideally be set just low enough to detect most target edges. Any additional edges detected in the scene only contribute to accumulator noise level, making it harder to detect true matches. In practice one cannot determine such a threshold level without having previously located the targets. A more useful rule is to set the threshold according to image type or previously measured edge density.

The number and density of edges detected in an image are sigmoid (s-shaped) functions of edge threshold similar to cumulative frequency histograms. GHOUGH operates best when 10% to 20% of the pixels are classified as edge points, although it will usually work well at any edge density above 6%. Some typical threshold values to

achieve specified edge densities are:

Scene Type	6%	12%	25%	50%
Cloudy sky	42	35	28	20
Aerial terrain	160	120	80	40
Aerial target area	200	180	120	60
Low-angle urban	260	200	140	90
Forest cover	280	220	160	100
Aerial urban	720	600	480	340

A more accurate determination may be made using the verbose option of GHOUGH to print out the edge density found at a given threshold. This test can be done on a small image area to choose a threshold for the entire image. (Future versions could use an adaptive threshold that varies with edge density in each portion of the image.)

Occasionally the position, rotation angle, or radius reported for a target will vary slightly as a function of edge threshold. This is because the spatial distribution of edges in an image may vary with the definition of what constitutes an edge. GHOUGH gives equal weight to all edge elements regardless of their relative strengths or continuity. This is both a strength and a weakness of the algorithm: it permits shapes to be found in very noisy imagery, but reports nonintuitive detections for smoothly varying intensity regions such as hot spots, specular reflections, rounded surfaces, and shorelines.

In general it is better to use too low a threshold: this will increase chances of finding target edges while only slightly increasing noise level, and the edges found are likely to be the most reliable ones. The main drawback is that low thresholds increase the time required to fill the accumulator with votes. A reasonable starting guess is a threshold of 120.

Contrast setting:

The user is also asked whether to search for dark (black) objects on light backgrounds, light (white) objects on dark backgrounds, or both (mixed). Specifying 'mixed' will increase execution time and slightly increase the accumulator noise level. Use the most restrictive specification that will do the job.

6.2. Performance Statistics

This section documents the performance of GHOUGH during Testbed evaluation trials. Formulas are given, where appropriate, but most of the information is subjective.

Edge processing time:

The processing time (in seconds) required to compute edge locations and increment the corresponding accumulator cells may be modeled as

Evaluation

$$\textit{Edge time} = .00036(\textit{window points}) + .0053(\textit{edges found})$$

$$+ .00019(\textit{accumulator entries}) + (\textit{additional paging time})$$

where

$$\textit{Edges found} = (\textit{window points})(\textit{edge density})$$

$$\textit{Accumulator entries} = (\textit{edges found})(\textit{entry density})$$

$$\textit{Entry density} = (\textit{accumulator planes})(\textit{contrasts})$$

$$(\textit{quantization factor})(\textit{border effect})$$

The quantization factor is unity for a circle or smoothly varying template (whether convex or involuted), and is approximately one eighth the number of template points for a square. The border effect, or percentage of votes falling within the accumulator, is more difficult to estimate. It is typically near unity, but may be less than .25 when a large target is sought in a small image window.

Window points, edges found, and the number of accumulator entries are all nearly linear predictors of each other, but the proportionality constants depend on edge density and entry density. The "edges found" coefficient is slightly dependent on template density and on rotation range because of the time required to locate the proper angle in the R-table. A fraction of the time for "accumulator entries" is due to instrumentation code inserted for the evaluation; the rest is largely spent computing cell addresses.

A "normal" amount of virtual memory accessing is included in the above formulas. For very large accumulators there will be substantial amounts of additional paging time. This time increases with accumulator volume, edges found, system load, and with various system parameters. It also increases with increasing template radius or rotation range because the votes for template centers "jump around" more in the accumulator. Involuted or angular template shapes may also cause paging for this reason.

An additional penalty due to this extra paging is not included in the above equation: "real time" spent waiting for page requests to be satisfied. If other jobs are running on the system, this time may not be wasted, but it will delay completion of the GHOUGH task. For very large accumulators (*e.g.*, two million cells), the real time on an unloaded system may be several times the processing time used.

Accumulator search time:

Let

$$\textit{Accumulator density} = \frac{\textit{accumulator entries}}{\textit{accumulator volume}}$$

be the average number of entries in an accumulator cell. For densities below 0.5 the time required to search the accumulator for local maxima is almost entirely that required to examine each cell once and to sort the resulting list of maxima. This is because few cells reach the threshold of three counts required for further processing. A good formula is

Evaluation

$$\textit{Analysis time} = .000042(\textit{search volume})+.0025(\textit{maxima found})$$

This time is typically small and nearly constant.

For accumulator densities above 0.5 the search time depends on the density. Two opposing tendencies may be at work: more cells with at least three counts are found, but, with increasing density, the average number of neighbors that must be checked decreases (since there is a greater chance that a given neighbor has a count greater than the original cell). The net result may be modeled as

$$\textit{Analysis time} = 10^{-4}(\textit{search volume})(.08+2.6\log(1+\textit{accumulator density}))$$

$$+(\textit{additional paging time})+.0025(\textit{maxima found})$$

where the logarithm is to the base 1 /. The additional paging time depends on accumulator size, compactness (*i.e.*, search volume/accumulator volume), system load, and other system parameters. It may exceed 60 seconds in some cases, but is usually much smaller than the paging time during edge processing.

Local maxima:

The number of local maxima in the accumulator search volume is dependent on the "noise statistics," or the distribution of false counts in the accumulator. For unit search grain and a convex template, the number of maxima having at least three counts is approximately

$$\textit{Maxima} = .023(\textit{search volume})^{0.8}(1+\textit{accumulator density})^2-1$$

This number ranges from none to more than 6000. (The reporting of multiple maxima was added to GHOUGH as part of the Testbed evaluation effort. Future versions may use an adaptive threshold to prevent such numbers of false "hits" from being reported.)

At coarser search grains, the number of maxima drops much faster than this formula brings it up. Increasing spatial coarseness in the accumulator increases the number of local maxima until the accumulator density is about 1.0, then decreases it until there is only one maximum (or none) in the accumulator. As a rough approximation, using a search grain of two may double or halve the number of maxima found with unit search grain; further halving of the resolution will drop the number of maxima by a factor of five.

An angular template will also produce accumulator densities high enough to invalidate this formula. Halving the search resolution will again cause the number of maxima to drop by a factor of five or more, with no special exemption for a search grain of two.

Noise threshold:

The highest cell count due to noise is related to the search volume since larger volumes mean more chances for coincidental clusters of votes. Scene content (other than edge density) has little effect on noise statistics. A reasonable model is

$$\textit{Noise} = 2.04(\textit{search volume})^{0.09}(1+\textit{accumulator density})^{0.8}-1$$

which is usually accurate to within one count for noise counts of ten or fewer. For

most purposes, the noise threshold is small and nearly constant, but use of coarse search grain can push it into the hundreds. This formula seems to work for angular templates, but oversampling of a circular template (producing severe quantization errors) can increase the noise level by 50%.

The number of "noise maxima" found at successive threshold levels tends to increase by a factor of five. Thus a single cell with ten votes will be accompanied by five cells with nine votes, 25 with eight votes, etc. This pattern is not particularly reliable, particularly at high accumulator densities, but it could be used as the basis for an adaptive threshold: increase the threshold and discard all lower local maxima whenever, say, 50 maxima have been found above the current threshold. A tighter limit could be used if the number of valid targets could be estimated.

Detection performance:

As yet there exists no quantitative model for the number of votes required to indicate a shape instance. Some of the factors controlling the maximum count due to a target match are: average accumulator noise count, template shape, quantization factor, density of template points compared to positional and orientational resolution, target radius, position of the target relative to the quantized spatial grid, width of the target border and gradient across it, and percentage of border edges detected.

The GHOUGH program does not currently offer a threshold for automatically screening good matches from bad ones. If one were installed, the proper setting would be just above (or perhaps just below) the computed noise threshold. Higher-level intelligence would still be needed to screen matches reported by GHOUGH.

GHOUGH match reliability has been estimated by searching for targets (circles, ellipses, and a lake outline) in various images that did, or did not, contain such targets. A target was considered to be "found" if GHOUGH reported one or more matches within 15% of the true size at a location close enough (again, 15% of the true radius) to the known position. Angular accuracy was not checked, nor was the cell count compared to any computed noise threshold.

In one series of tests, the program searched for twelve small, nearly circular ellipses. The aspect ratio, rotation, and radius were known, although some searches permitted a small range for each parameter. The search window was either 128x128 or 254x254, and the "known" positions of the twelve targets were defined within each of these search windows (even for images without targets at these positions). A true "hit" or a false match (depending on the image) was reported if the "known" targets were among the first 20 local maxima.

False matches in images not containing targets were rare: no more than one of the "known" targets was ever reported, and these false matches had vote counts within the noise level. The false matches occurred in a view of urban San Francisco with the edge detector threshold set for very high edge densities (.45 to 1.00).

True matches and missed targets can only be reported for the scene containing the targets. For most runs this was the *plant* picture provided with the original GHOUGH program. It was very difficult to locate all twelve elliptical storage tanks in the plant image, probably because the edges are blurred and the radius is only 7 pixels. Detection of all twelve required a search grain of 2 as well as a critical edge detector threshold. At unit search grain, eleven of the twelve could be found at edge threshold 120, and eight or more were found with any threshold of zero to 180 (edge density 1.00 to

.10).

The above tests involved searching for dark objects on light backgrounds. Setting the contrast parameter to 'mixed' doubled the accumulator vote density, with a corresponding increase of a few counts in the noise level (*e.g.*, from six to seven). This did not prevent finding eleven of the twelve storage tanks (at the optimum edge density) since they had as many as 14 votes. The more general search also turned up a few white objects against dark backgrounds that were quite reasonable "hits."

Circles were also sought in various aerial images containing cylindrical storage tanks. Several images of Fort Belvoir showed multiple white storage tanks ranging from two to 19 pixels in radius. These tanks were extremely obvious to a human observer, but the GHOUGH program had great difficulty in finding them. Various parameter combinations were tried, but some targets resisted detection even when the search was tightly constrained to exactly the right location and size. Further, it was impossible to predict from visual appearances which of the tanks would be difficult to find; neither size nor contrast against background was a good criterion.

For example: the *ftb1a* image contains 29 clearly visible storage tanks in three clusters. The circular (or dome) tank tops range from two to six pixels in radius. A file was constructed giving the true location and radius of the tank tops as presented in this image. Attempts to find all of the tanks in one run proved futile. Some tanks were missed in each cluster, and one cluster was particularly difficult to match.

The obvious next step was to search through each cluster independently. This did improve detection probabilities, but results were still disappointing. One representative search over a cluster of 14 tanks located only three of the targets (to within 15% of true radius) in the first 40 matches. These matches were accompanied by numerous "almost" matches and a few wild "hits" that were difficult to perceive even after they were pointed out.

The *ftb1b* image is an enlargement of this cluster of 14 storage tanks. The circular tops range from eight to 19 pixels in radius. A search similar to that above was done using unit search grain and unit resolution in radius to find white circular objects. This time the "score" function reported multiple matches on six of the 14 tanks. Another three tanks were almost matched in the first 40 local maxima, but either the ground truth file was slightly inaccurate or the 15% scoring tolerance was too tight for the score function to report them. The remaining five tanks were matched very badly or not at all.

There is no reason to believe that software errors were responsible for the poor performance. A reasonable explanation involves the cylindrical nature of the targets: since they were viewed off-axis, they presented a roughly elliptical aspect consisting of the circular top overlapping the circular base. The accumulator thus gathered conflicting evidence for circles matching the top, the base, and various approximations to the elliptical composite, all with quantization errors due to the full-pixel spatial resolution. The spatial and orientational search grains were coarse enough that only one of these shapes could be reported, and very often the "wrong" shape had the most support.

Two improvements to the program might have prevented this situation: allow subpixel spatial resolution and analyze each accumulator plane independently so that existence of one circle does not necessarily imply nonexistence of similar circles. These and other suggestions are elaborated below. For the current implementation, however, the lesson is that the set of targets reported by the program must be

screened and edited by a more intelligent system. GHOUGH may assist a photointerpreter, but cannot yet replace him.

Section 7

Suggested Improvements

The process of evaluation has turned up numerous ways to improve the current GHOUGH implementation. Comments about existing features have been made at the appropriate points throughout this document. The following are additional suggestions for substantial modifications or needed research. Some of these would require major research projects or are beyond the scope of the original program. (The large number of suggestions should not be taken as a criticism of the GHOUGH system. Rather it is a tribute that the approach is flexible enough to support such extensions and is promising enough to be worth the effort.)

* *Linear Templates*

  Evaluation of GHOUGH as a corner detector was done using a right-triangle template having 20 points along each for the shorter sides and no points along the hypotenuse. This prevented the diagonal side from having a significant effect on the analysis, but a better solution should be implemented. The current restriction to closed templates is unnecessary. A simple modification would permit line segments, corners, riverbanks, and other fragments to be located. Multiregion templates (*e.g.*, outlines of machine parts with holes) should also be permitted.

* *Composite Templates*

  The current GHOUGH program searches for only one simple shape at a time. The principal author [Ballard81a] has suggested ways of searching for composite shapes, such as vehicles with visible wheels. One method uses the union or difference of R-tables to search for a particular composite. Another uses subtemplates independently to increment a common accumulator and then combines the evidence by convolving the accumulator with a special composite mask. These methods are nearly optimal for finding predictable, unarticulated shapes in low-quality imagery.

* *Subtemplate Accumulators*

  In higher quality imagery it may be better to search for the subtemplates independently using different accumulators and then invoke higher-level logic to combine the evidence. This permits sophisticated scene analysis, but the large number of generalized Hough analyses would consume both memory and computer time. Specialized Hough line, circle, and corner detectors should therefore be used.

Suggested Improvements

* *Improved Template Entry*

  The current method of entering templates using a keyboard-controlled cursor is clumsy. The program should be interfaced to a digitizing tablet or other tracing device. Tracing should be integrated with zoom capability to increase accuracy and reduce quantization errors. There should also be retrace and interpolation commands to simplify improvement of existing coarse templates. Entry of boundary points with subpixel accuracy and reliable edge angles should be supported.

* *Automated Template Extraction*

  Search performance could be improved by extracting template edge element directions from the image during initial tracing. The user would indicate an approximate boundary point, and the system would identify the nearest good boundary point and the gradient direction at that point. The increased accuracy of the edge positions and the edge directions at those points would reduce the number of boundary points needed to obtain a given search accuracy. This, in turn, would reduce execution time. A related capability, particularly useful for tracking applications, would allow a new template to be extracted automatically after GHOUGH had registered an existing template with the image.

* *Gradient Map Input*

  The current implementation of edge detection is too slow. The program selects a pixel position, reads in the neighborhood data, computes the edge gradient, and, possibly, increments the accumulator. It then moves to the next pixel and begins again with an overlapping neighborhood. This is inefficient, particularly with neighborhood windows larger than 3x3. For most applications it would be more efficient to use a separate moving-window gradient operator to provide gradient or edge map input to the GHOUGH program. The same map could then be used to locate many different objects using separate runs of GHOUGH or other programs.

* *Adaptive Edge Threshold*

  The user must set a threshold value for choosing which gradients are strong enough to be useful. Too low a threshold will introduce noise and slow the computation; too high a threshold will miss low-contrast edges. Future versions of the program should use adaptive thresholds instead of this constant global threshold.

* *Adaptive Edge Resolution*

  The current edge detector might also be improved. The 3x3 Sobel edge detector is excellent for finding small objects with sharp edges, but is nonoptimal for large objects and gradual edges. Use of different detector sizes requires separate GHOUGH programs to be run, and there is no way to combine evidence from multiple runs. The best solution would be a hierarchy of

edge detectors of different sizes all applied to each point in the image. The edges found at different resolutions could each be allowed to vote for shapes, or the single edge with the strongest support could be allowed to vote. It might also be possible to incorporate line detectors, corner detectors, and edge curvature measures for even better performance.

* *Confidence Weighting*

Each detected edge element votes once for each shape that could contain it, and each edge element has equal importance regardless of the strength of the edge. It may be desirable, as a user-selected option, to assign more weight to the edge elements that are more certain [O'Gorman73, van Veen81].

* *Negative Voting*

The problem of finding local maxima in the accumulator is necessitated by the use of positive weights for all votes. If the voting pattern were zero-mean, incorrect (or noise) cells would tend toward zero counts while true peaks would continue to accumulate positive votes. This technique, called CHOUGH or Complementary Hough, has recently been investigated by Brown at the Univ. of Rochester [Brown82a, Brown82b]. He recommends that each positive vote be accompanied by negative votes of 1/2 on each side across the direction of the voting outline. This represents the fact that positive evidence for one shape is also negative evidence for other similar shapes. Brown has also developed an analogy between GHOUGH voting patterns and standard analytic techniques in optics. (CHOUGH voting patterns are similar to the diffraction patterns of coherent optics, and also to the lateral inhibition patterns present in neurological vision systems.)

* *Sparse Edge Detector Application*

Coarse search is currently implemented by combining cells in the accumulator. This greatly reduces computer memory requirements but only slightly reduces execution time. An additional saving could be achieved by applying the edge operator at every nth row and column to match the spatial grain of the accumulator. This saving would be small if the edge operator were still applied at full resolution.

* *Multiple Resolution Analysis*

An even simpler approach to coarse search would use a "pyramid" of images at different resolutions. This would eliminate the need for a search grain increment in applying edge operators and indexing the accumulator. The program would start with a highly reduced image (or with an edge map of such an overview image) and would attempt to find any objects visible at that resolution. It would then use a local search in a higher resolution image to confirm and precisely locate the objects. (A disadvantage of any coarse search method is that, once an object is missed, it may never be found at higher resolutions.)

Suggested Improvements

* *Arbitrary Windows*

The current restriction to rectangular search areas is due to the implementation of the accumulator as an array registered with the image window. There are other implementations, perhaps more expensive, which would allow edge-search windows of arbitrary shape. At the least it should be possible to specify multiple rectangular windows instead of having to run the GHOUGH search separately for each area or jointly with a much larger accumulator.

* *Independent Search Areas*

In the original GHOUGH implementation, the user was asked to specify a shape-center search area within the edge-search window. This helped maintain resolution when estimating the position of a large object in a slightly larger edge-search window. If this option is reinstated, the center search window should not be restricted to be within the edge search window, or even within the image.

* *Arbitrary Resolution*

The spatial resolution is currently limited to whole pixels, and the angular resolution is limited to whole degrees. The radius scale factor, on the other hand, may be specified to any desired resolution. There is no inherent reason why all four parameters should not be kept in floating-point, thus permitting very fine shape matching when the approximate match position is already known. Edge detectors with subpixel accuracy (*e.g.*, [MacVicar-Whelan81]) could also be used for fine searches.

* *Storage Reduction*

Greater spatial and orientational accuracy are possible if the accumulator uses fewer bits per cell. The accumulator now consists of 32-bit integers, but should be reduced to 16-bit integers. The highest count encountered in numerous runs was 678, obtained with a square template applied to an urban scene with a search grain of 16 in each direction. At unit search grain, the highest counts rarely exceed 40, or even 20. A hardware implementation might manage with five bits per accumulator cell, or perhaps with even fewer by using logarithmic counts with stochastic updating [Morris78].

* *Logarithmic Radius Spacing*

The specification of radius range by a minimum, maximum, and increment makes it very difficult to maintain resolution while covering a useful range. The spacing between sequential radius values should be logarithmic instead of incremental. With certain accumulator analysis schemes, the search grain could also be made proportional to radius.

Suggested Improvements

* *Adaptive Accumulator*

  Another method for increasing the resolution is to use an adaptive accumulator [O'Rourke81a, Sloan81]. Such an accumulator is continually restructured as the votes are entered so that high resolution is maintained in those areas that are receiving many votes. This would permit precise matching in a single pass instead of using a coarse search followed by a fine search. The cost in algorithm complexity might be quite high, however.

* *Hash-Table Accumulator*

  An attractive alternative to a full adaptive accumulator is one based on a hash algorithm. Only those cells that have significant counts are maintained; others are flushed periodically or as accumulator space is needed. Although the best accumulator size and flushing strategies are not yet known, Brown reports encouraging results [Brown82c].

* *Accurate R-Table Indexing*

  The current algorithm for using the R-table always rounds angles up to the next larger table entry. This is acceptable for many purposes, but disastrous with an angular template such as a square or right angle. The algorithm should be modified to round to the nearest table entry. This improvement could be combined with a change in the R-table format; a hash table or ragged array representation could be referenced faster than the current linked-list format.

* *Efficient Accumulator Indexing*

  The accumulator is currently accessed by column, row, rotation, and radius, with the radius subscript varying most quickly. A more efficient subscripting order might be found, although this one seems reasonable. A sizable improvement in paging performance might be possible using Quam's method of four-dimensional block storage [Quam80]. This would reduce the number of accumulator entry page faults caused by cycling the rotation and radius associated with each image edge element. The efficiency gained would depend on the number of storage words in each virtual memory page.

* *Elimination of Accumulator Padding*

  The accumulator search should be modified so that points outside the search volume are never examined. This will increase the accumulator search time and search complexity, but the effect will be small. In return the accumulator volume and the number of entries into the accumulator will be greatly reduced, sometimes by a factor of nine. This will also reduce system paging time, which can be a major component of the total run time.

Suggested Improvements

* *Accumulator Save/Restore*

GHOUGH allows individual planes of the accumulator to be written out as images. This capability should be augmented so that the entire accumulator could be written out and later restored. Researchers could then test various accumulator analysis algorithms without having to repeat the edge detection and accumulator building steps. There should be a similar capability for saving and restoring the list of matches. These modifications suggest a slightly different user interface than currently exists; one where the setting or modification of search parameters is a separate step from the activation of routines using those parameters.

* *Adaptive Match Screening*

Too many potential matches (*e.g.*, over 6000) are often reported, partly because of lack of smoothing (see below). The program should discard poor matches when much better ones are found. There seems to be no reason for reporting more than a few hundred, or even a few dozen, potential matches. The required screening could be made automatic, or could be controlled by user-settable parameters.

* *Accumulator Smoothing*

Votes for a single instance of a large shape tend to be reported as multiple maxima (when using the 3x3 edge detector). Both the spreading of the true peak in the accumulator and the reporting of false maxima due to noise can be combatted by blurring the four-dimensional accumulator values before attempting to find local maxima. The program authors recognize the need for such blurring [Sloan80, Ballard81a], but do not suggest any particular convolution function. A simple center-weighted 3x3x3x3 mask may be sufficient.

* *Relaxation Enhancement*

The smoothing suggested above may exacerbate the problem of one shape instance suppressing the detection of other similar instances (*e.g.*, overlapping circles in images of storage tanks). Either simpler or more complex relaxation and reinforcement processes might be more effective for particular applications.

* *Cluster Analysis*

Another possibility would be to search for maxima independently in each accumulator plane and then combine the resulting lists. This might offer great savings in accumulator analysis time if the planes used were aligned with the data storage order. The current "score" routine recognizes matches as being equivalent if they are within tolerance of a known target. A more sophisticated algorithm could determine clusters of reported matches without knowing ground truth.

Suggested Improvements

* *Match Quality Measures*

  At present the only information reported about a match is the number of votes it received. The reporting algorithm could also determine peak sharpness, four-dimensional volume, parameters of the best fitting paraboloid, peak-to-sidelobe ratio, or other descriptive statistics. These could be used by a higher-level program to screen or combine the reported matches and to estimate the true position in the accumulator-to-subcell accuracy. The quality of a match might also be estimated from the cell count since each cell has a theoretical maximum count that is dependent on the noise level and the completeness of the corresponding shape instance.

* *Match Verification*

  Future versions of the GHOUGH program may also project matches back into the image to check for continuity [Kitchen81] or to use a warp-tolerant match verification such as chamfer matching [Barrow77] or elastic matching [Burr81]. Higher-level knowledge such as the plausibility of occluding objects [Chien74] or of particular target configurations [Fischler73, Brown79, Price81] might be used for validation.

* *Multiframe Validation*

  Another possibility is multiframe validation [Iler79, McIngvale80]. A reliable match will be consistently reported, whereas an unreliable match will be reported at different locations in different frames. The frame-to-frame statistics of a match may thus be used to determine match quality. The number of frames that can be used depends on the rate at which frames are matched, the speed with which registration positions normally change, and the available time for target lock-on.

* *Fuzzy Display*

  GHOUGH displays each detected match as a sharp shape instance corresponding to the accumulator cell indices of the local maxima. The spatial and orientational quantization of the accumulator makes this a misleading representation. The shape could be displayed as a somewhat fuzzy overlay indicating the ambiguity in true target location. A better solution, of course, would be to locate the target more precisely either by interpolation in the accumulator or by match verification in the image.

* *Improved Score Function*

  The command that compares detected matches to expected target locations should similarly be modified to take search resolution into account. It makes no sense to reject a match with a four-pixel error if the search grain is eight pixels.

Suggested Improvements

* *Improved User Interface*

  The GHOUGH program allows the user to examine the list of detected targets using several screening criteria. Two extensions are needed: (1) storage and later recovery of the list for delayed analysis and (2) additional screening routines based on radius, orientation, match quality, or arbitrary combinations of factors. A separate command language or editing system should be developed for working with these lists. A LISP language might make a good basis.

* *Artificial Intelligence*

  Feature-matching systems have always been limited by the quality of the feature-extraction process. Future systems may integrate feature extraction with feature matching so that high-level considerations can direct the application of low-level image operators [Perkins73]. The high-level analysis may even direct the gathering of additional imagery. This falls within the planning and expert system areas of artificial intelligence [Garvey74, Garvey76, Brown79].

Section 8

Conclusions

The GHOUGH program is an efficient and flexible template-matching system. Its greatest potential is in cueing, counting, and mensuration applications. Its weaknesses are mainly correctable, although the inherent quantization of the Hough accumulator space does limit accuracy on any single pass through an image.

The current implementation requires a large number of user-supplied parameters in order to limit the search space. This is inconvenient for simple interactive use, but necessary given current computer resources. The ability to focus on specific recognition problems will be of more use if generalized Hough techniques are built into sophisticated or intelligent systems.

We have documented the GHOUGH program, evaluated its performance, and suggested ways to improve the algorithm and its implementation. Analytic models have been provided where appropriate. While the existing system has definite limitations, it demonstrates many promising uses for the GHOUGH algorithm.

Implementation of the GHOUGH program on the DARPA IU Testbed required extensive modification of the user interface and had considerable influence on the Testbed itself. The merging of the GHOUGH code from the University of Rochester with graphics subroutines and the CI command interpreter from Carnegie-Mellon University was a validation of the Testbed concept.

References

[Arcelli71] C. Arcelli and S. Levialdi, "Picture Processing and Overlapping Blobs," *IEEE Trans. on Computers*, pp. 1111-1115, Sep. 1971.

[Baker80] H.H. Baker, "Edge Based Stereo Correlation," *Proc. Image Understanding Workshop*, College Park, MD, pp. 168-175, Apr. 1980.

[Ballard76] D.H. Ballard and J. Sklansky, "A Ladder-Structured Decision Tree for Recognizing Tumors in Chest Radiographs," *IEEE Trans. Comput.*, Vol. C25, pp. 503-513, 1976.

[Ballard81a] D.H. Ballard, "Generalizing the Hough Transform to Detect Arbitrary Shapes," *Pattern Recognition*, Vol. 13, No. 2, pp. 111-122, 1981.

[Ballard81b] D.H. Ballard, "Parameter Networks: Towards a Theory of Low-Level Vision," *Proc. 7th Int. Jnt. Conf. on Artificial Intelligence (IJCAI-81)*, Vol. 2, pp. 1068-1078, 1981.

[Ballard81c] D.H. Ballard and D. Sabbah, "On Shapes", *Proc. 7th Int. Jnt. Conf. on Artificial Intelligence (IJCAI-81)*, Vol. 2, pp. 307-612, 1981.

[Barrow77] H.G Barrow, J.M. Tenenbaum, R.C. Bolles, and H.C. Wolf, "Parametric Correspondence and Chamfer Matching: Two New Techniques for Image Matching," *Proc. Image Understanding Workshop*, pp. 21-27, Apr. 1977.

[Bastian71] P. Bastian and L. Dunn, "Global Transformations in Pattern Recognition of Bubble Chamber Photographs," *IEEE Trans. on Computers*, Vol. C-20, p. 995, Sep. 1971.

[Belsher79] J.F. Belsher, H.F. Williams, and R.H. Kin, "Scene Matching with Feature Detection," *Digital Processing of Aerial Images*, SPIE Vol. 186, pp. 12-20, 1979.

[Binford81] T.O. Binford, "Spatial Understanding," *Proc. Image Understanding Workshop*, Washington, D.C., pp. 236-240, Apr. 1981.

[Bolles81] R.C. Bolles and M.A. Fischler, "A RANSAC-Based Approach to Model Fitting and its Application to Finding Cylinders in Range Data," *Proc. 7th Int. Jnt. Conf. on Artificial Intelligence (IJCAI-81)*, Vancouver, pp. 637-643, 1981.

[Bolles82] R.C. Bolles and R.A. Cain, *Recognizing and Locating Partially Visible Objects: The Local-Feature-Focus Method*, SRI Int., Tech. Note 262, Mar. 1982.

[Brown79] C.M. Brown, K.A. Lantz, and D.H. Ballard, "Model Driven Vision using Procedure Description," *U. of Rochester Computer Sci. Engineering Review*, pp. 16-24, 1978-79.

[Brown82a] C.M. Brown, *Bias and Noise in the Hough Transform I: Theory*, TR 105, Computer Science Dept., Univ. of Rochester, June 1982. A summary has been submitted to *IEEE Trans. on Pattern Analysis and Machine Intelligence*. as *Inherent Bias and Noise in the Hough Transform*.

[Brown82b] C.M. Brown, *Bias and Noise in the Hough Transform II: Experiments*, TR 113, Computer Science Dept., Univ. of Rochester, Aug. 1982.

[Brown82c] C.M. Brown and D.B. Sher, "Modeling the Sequential Behavior of Hough Transform Schemes," *Proc. Image Understanding Workshop*, Stanford, pp. 115-123, Sep. 1982.

[Burr81] D.J. Burr, "Elastic Matching of Line Drawings," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-3, No. 6, pp. 708-713, Nov. 1981.

[Chien74] R.T. Chien and Y.H. Chang, "Recognition of Curved Objects and Object Assemblies," *Proc. 2nd Int. Jnt. Conf. on Pattern Recognition*, Copenhagen, pp. 496-510, Aug. 1974.

[Chien75] Y.P. Chien, "On the Optimal Extraction of Boundary Curves," *IEEE Conf. on Computer Graphics, Pattern Recognition, and Data Structure*, Los Angeles, pp. 208-209, May 1975.

[Choate79] W.C. Choate and W.W. Boyd, "Correlation Tracking Concepts for THASSID," *Proc. IEEE 1979 Nat. Aerosp. and Electron. Conf. (NAECON 1979)*, Dayton, OH, pp. 79-85, May 1979.

[Clark79] C.S. Clark, W.O. Eckhardt, C.A. McNary, R. Nevatia, K.E. Olin, and E.M. VanOrden, "High-Accuracy Model Matching for Scenes Containing Man-Made Structures," *Digital Processing of Aerial Images*, SPIE Vol. 186, pp. 54-62, 1979.

[Davis82] L.S. Davis, "Hierarchical Generalized Hough Transforms and Line-Segment Based Generalized Hough Transforms," *Pattern Recognition*, Vol. 15, No. 4, pp. 277-285, 1982.

[Deal79] B. Deal, C.M. Lo, R. Taylor, V. Norwood, H. Henning, T. Daggett, T. Noda, J. Powers, G. Guzman, H. Greenberger, G. Towner, and G. Parker, *Automatic Target Cuer First Quarter Report*, Northrop Corp., Electro-Mechanical Division, Anaheim, CA, Report NORT-79Y100, 101 pp., Oct. 1979.

[Deans81] S.R. Deans, "Hough Transform from the Radon Transform," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-3, No. 2, pp. 185-188, Mar. 1981.

[Doudani77] S.A. Dudani and A.L. Luk, "Locating Straight-Line Edge Segments on Outdoor Scenes," *Proc. IEEE Conf. on Pattern Recognition and Image Processing*, pp. 367-377, 1977.

[Dougherty79] L.S. Dougherty, "A New Technique for Tracking Sequences of Digitized Images," *Proc. IEEE Conf. on Decision and Control*, Fort Lauderdale, FL, pp. 1028-31, December 1979.

[Duda72] R.O. Duda and P.E. Hart, "Use of the Hough Transform to Detect Lines and Curves in Pictures," *CACM*, Vol. 15, No. 1, pp. 11-15, 1972.

[Fischler73] M. Fischler and R.A. Elschlager, "Representation and Matching of Pictorial Structures," *IEEE Trans. on Computers*, Vol. C-22, No. 1, pp. 67-92, Jan. 1973.

[Fitts79] J.M. Fitts, "Precision Correlation Tracking via Optimal Weighting Functions," *Proc. IEEE Conf. on Decision and Control*, Fort Lauderdale, FL, pp. 280-283, December 1979.

[Flachs78] G.M. Flachs, P.I. Perez, R.B. Rogers, S.J. Szymanski, J.M. Taylor, and Y.H. U, *Real-Time Video Tracking Concepts*, New Mexico State University, Las Cruces, NM, Report NMSU-TR-78-1, May 1978, 69 pp.

[Flachs79] G.M. Flachs, P.I. Perez, R.B. Rogers, S.J. Szymanski, J.M. Taylor, and Y.H. U, *A Real-Time Video Tracking System*, New Mexico State University, Las Cruces, NM, January 1979, 180 pp.

[Freeman74] H. Freeman, "Computer Processing of Line-Drawing Images," *Computing Surveys*, Vol. 6, No. 1, pp. 57-97, Mar. 1974.

[Garvey74] T.D. Garvey and J.M. Tenenbaum, "On the Automatic Generation of Programs for Locating Objects in Office Scenes," *Proc. 2nd Int. Jnt. Conf. on Pattern Recognition*, Copenhagen, pp. 162-168, Aug. 1974.

[Garvey76] T.D. Garvey, "An Experiment with a System for Locating Objects in Multisensory Images," *Proc. 3rd. Int. Jnt. Conf. on Pattern Recognition*, Coronado, CA, pp. 567-575, Nov. 1976.

[Gennery77] D.B. Gennery, "A Stereo Vision System," *Proc. Image Understanding Workshop*, Palo Alto, pp. 31-46, Oct. 1977.

[Goodman79] I.R. Goodman, "A General Model for the Multiple Target Correlation and Tracking Problem," *Proc. 18th IEEE Conf. on Decision and Control*, Fort Lauderdale, FL, pp. 383-8, December 1979.

[Grimson80] W.E.L. Grimson, "Aspects of a Computational Theory of Human Stereo Vision," *Proc. Image Understanding Workshop*, College Park, MD, pp. 128-149, Apr. 1980.

[Griffith70] A.K. Griffith, *Computer Recognition of Prismatic Solids*. Ph.D. Thesis, Dept. of Math., MIT, June, 1970.

[Haralick75] R.M. Haralick, "A Resolution Preserving Textural Transform for Images," *Proc. Computer Graphics, Pattern Recognition, and Data Structure*, Los Angeles, pp. 51-61, May 1975.

[Hemami70] H. Hemami, R.B. McGhee, and S.R. Gardener, "Towards a Generalized Template Matching Algorithm for Pictorial Pattern Recognition," *Proc. IEEE Symposium on Adaptive Processes (9th) Decision and Control*, Austin, TX, 4 pp., Dec. 1970.

[Hinton81] G.E. Hinton, "A Parallel Computation that Assigns Canonical Object-Based Frames of Reference," *Proc. 7th Int. Jnt. Conf. on Artificial Intelligence (IJCAI-81)*, Vol. 2, pp. 683-685, 1981.

[Hough62] P.V.C. Hough, *Method and Means for Recognizing Complex Patterns*, U.S. Patent 3069654, 1962.

[Hwang79] J.J. Hwang, C.C. Lee, and E.L. Hall, "Locating Vertices in a Scene," *Proc. IEEE SOUTHEASTCON*, pp. 357-362, 1979.

[Iannino78] A. Iannino and S.D. Shapiro, "A Survey of the Hough Transform and Its Extensions for Curve Detection," *Proc. IEEE Conf. on Pattern Recognition and Image Processing*, Chicago, pp. 32-38, May 1978.

[Iler79] T.W. Iler, M. Pitruzzello, and P.H. McIngvale, "Design and Evaluation of an Automatic Hand-Off Correlator," *Wkshp. on Imaging Trackers and Auton. Acq. App. for Missile Guidance*, Redstone Arsenal, AL, pp. 310-24, November 1979.

[Kimme75] C. Kimme, D.H. Ballard, and J. Sklansky, "Finding Circles by an Array of Accumulators," *CACM*, Vol. 18, pp. 120-122, 1975.

[Kitchen81] L. Kitchen and A. Rosenfeld, "Edge Evaluation using Local Edge Coherence," *Proc. Image Understanding Workshop*, Washington, D.C., pp. 99-113, Apr. 1981.

[Klein77] G.M. Klein and S.A. Doudani, "Locating Man-Made Objects in Low-Resolution Outdoor Scenes," *Applic. of Digital Image Processing*, SPIE Vol. 119, pp. 278-283, 1977.

[Laws77] K.I. Laws, "Circle Detection in Noisy Images," *Semiannual Technical Report*, USCIPI 770, Image Processing Institute, U. of Southern California, Los Angeles, Sep. 1977.

[Lowe80] D.G. Lowe, "Solving for the Parameters of Object Models from Image Descriptions," *Proc. Image Understanding Workshop*, College Park, MD, pp. 121-127, Apr. 1980.

[Lucey76] D.J. Lucey, G.E. Forsen, and M.J. Zoracki, *Processing of FLIR Data on DICIFER*, Pattern Analysis and Recognition Corp., Rome, NY, Report PAR-76-26, 108 pp., July 1976.

[MacVicar-Whelan81] P.J. MacVicar-Whelan and T.O. Binford, "Line Finding with Subpixel Precision," *Proc. Image Understanding Workshop*, Washington, D.C., pp. 26-31, Apr. 1981.

[Martelli76] A. Martelli, "An Application of Heuristic Search Methods to Edge and Contour Detection," *CACM*, Vol. 19, No. 2, pp. 73-83, Feb. 1976.

[Maybeck79] P.S. Maybeck and D.E. Mercier, "A Target Tracker using Spatially Distributed Infrared Measurements," *Proc. 18th IEEE Conf. on Decision and Control*, Fort Lauderdale, FL, pp. 285-9, December 1979. Also pub. in *IEEE Trans. Auto. Control*, Vol. AC-25, No. 2, pp. 222-5, April 1980.

[McIngvale80] P.H. McIngvale and M.C. Pitruzzello, *A Summary of the Development and Evaluation of the Automatic Target Handoff Correlator*, Army Missile Command, Redstone Arsenal, AL, Report RG-80-17, 65 pp., Jan. 1980.

[McKee76] J.W. McKee and J.K. Aggarwal, "Computer Recognition of Partial Views of Three Dimensional Curved Objects," *Proc. 3rd Int. Jnt. Conf. on Pattern Recognition*, Coronado, CA, pp. 499-503, Nov. 1976.

[McQueen81] M.P.C. McQueen, "A Generalization of Template Matching for Recognition of Real Objects," *Pattern Recognition*, Vol. 13, No. 2, pp. 139-145, 1981.

[Merlin75] P.M. Merlin and D.J. Farber, "A Parallel Mechanism for Detecting Curves in Pictures," *IEEE Trans. on Computers*, Vol. C24, pp. 96-98, 1975.

[Milgram78a] D.L. Milgram, A. Rosenfeld, T. Willet, and G. Tisdale, *Algorithms and Hardware Technology for Image Recognition*, Maryland University Computer Science Center, College Park, MD, 287 pp., March 1978.

[Milgram78b] D.L. Milgram, "Edge Point Linking using Convergent Evidence," *Proc. Image Understanding Workshop*, pp. 85-91, Nov. 1978.

[Morris78] R. Morris, "Counting Large Numbers of Events in Small Registers," *Communications of the ACM*, Vol. 21, No. 10, pp. 840-842, Oct. 1978.

[Nahi78] N.E. Nahi and S. Lopez-Mora, "Estimation-Detection of Object Boundaries in Noisy Images," *IEEE Trans. on Automatic Control*, Vol. AC-23, No. 5, pp. 834-846, Oct. 1978.

[Narendra79] P.M. Narendra and J.J. Grabau, *Advanced Pattern Matching Techniques for Autonomous Acquisition*, Honeywell Systems & Research Center, Minneapolis, MN, Report 79SRC104, December 1979, 62 pp.

[Nevatia76] R. Nevatia, "Locating Object Boundaries in Textured Environments," *IEEE Trans. on Computers*, Vol. C-25, No. 11, pp. 1170-1175, Nov. 1976.

[Noges80] E. Noges, A.M. Savol, A.J. Witsmeer, and D. Moerdyke, "Application of an Iterative Feature Matching Algorithm to Terminal Homing," *Image Processing for Missile Guidance*, SPIE Vol. 238, pp. 243-254, 1980.

[O'Gorman73] F. O'Gorman and M.B. Clowes, "Finding Picture Edges through Collinearity of Feature Points," *Proc. 3rd Int. Jnt. Conf. on Artificial Intelligence*, pp. 543-555, 1973.

[O'Rourke81a] J. O'Rourke, "Dynamically Quantized Spaces for Focusing the Hough Transform," *Proc. 7th Int. Jnt. Conf. on Artificial Intelligence (IJCAI-81)*, Vol. 2, pp. 737-739, 1981.

[O'Rourke81b] J. O'Rourke, "Motion Detection using Hough Techniques," *IEEE Conf. on Pattern Rec. and Image Processing*, Dallas, pp. 82-87, Aug. 1981.

[Orton79] D.A. Orton and G.N. Yutzi, "The AI2S Microprocessor Based Multimode Tracker," *Wkshp. on Imaging Trackers and Auton. Acq. App. for Missile Guidance*, Redstone Arsenal, AL, pp. 175-91, November 1979.

[Otazo79] J.J. Otazo and R.R. Parenti, "Digital Filters for the Detection of Resolved and Unresolved Targets Embedded in Infrared (IR) Scenes," *Smart Sensors*, Washington, D.C., Proc. SPIE, Vol. 178, pp. 13-24, April 1979.

[Perkins73] W.A. Perkins and T.O. Binford, "A Corner Finder for Visual Feedback," *Computer Graphics and Image Processing*, Vol. 2, pp. 355-376, 1973.

[Perkins76] W.A. Perkins, "Multilevel Vision Recognition System," *Proc. 3rd Int. Jnt. Conf. on Pattern Recognition*, Coronado, CA, pp. 739-744, Nov. 1976.

[Price75] K. Price and R. Reddy, "Change Detection in Multi-Sensor Images," *Proc. 10th Int. Symp. on Remote Sensing of the Environment*, Ann Arbor, MI, pp. 769-776, Oct. 1975.

[Price81] K.E. Price, "Relaxation Matching Applied to Aerial Images," *Proc. Image Understanding Workshop*, Washington, D.C., pp. 22-25, Apr. 1981.

[Pridgen78] J.H. Pridgen, W.W. Boyd, W.C. Choate, E.E. Mooty, and R.B. Ottesen, "Terminal Homing Applications of Solid-State Imaging Devices (THASSID) Composite Tracking Concepts," *Digital Processing of Aerial Images*, Huntsville, AL, Proc. SPIE, Vol. 186, pp. 73-87, May 1979.

[Quam80] L.H. Quam, "A Storage Representation for Efficient Access to Large Multi-Dimensional Arrays," *Proc. Image Understanding Workshop*, College Park, MD, pp. 104-111, Apr. 1980.

[Reid79] D.B. Reid, *The Application of Multiple Target Tracking Theory to Ocean Surveillance*, Lockheed Palo Alto Research Lab., Palo Alto, CA, 1979.

[Rosenfeld69] A. Rosenfeld, *Picture Processing by Computer*. Academic Press, New York, 1969.

[Rosenfeld78] A. Rosenfeld, "Iterative Methods in Image Analysis," *Pattern Recognition*, Vol. 10, No. 3, pp. 181-7, 1978.

[Sabbah81] D. Sabbah, "Design of a Highly Parallel Visual Recognition System", *Proc. 7th Int. Jnt. Conf. on Artificial Intelligence (IJCAI-81)*, Vol. 2, pp. 722-727, 1981.

[Shapiro74] S.D. Shapiro, "Detection of Lines in Noisy Pictures using Clustering," *2nd Int. Jnt. Conf. on Pattern Recognition*, Copenhagen, pp. 317-318, Aug. 1974.

[Shapiro75] S.D. Shapiro, "Transformations for the Computer Detection of Curves in Noisy Pictures," *Computer Graphics and Image Processing*, Vol. 4, pp. 328-338, 1975.

[Shapiro76] S.D. Shapiro, "An Extension of the Transform Method of Curve Detection for Textured Image Data," *3nd Int. Jnt. Conf. on Pattern Recognition*, Coronado, CA, pp. 205-207, Nov. 1976. See also *IEEE Trans. on Computers*, Vol. C-27, p. 254, Mar. 1978.

[Shapiro78a] S.D. Shapiro, "Feature Space Transforms for Curve Detection," *Pattern Recognition*, Vol. 10, pp. 129-143, 1978.

[Shapiro78b] S.D. Shapiro, "Generalization of the Hough Transform for Curve Detection in Noisy Pictures," *Proc. 4th Int. Jnt. Conf. on Pattern Recognition*, Kyoto, Japan, pp. 710-714, Nov. 1978.

[Shapiro78c] S.D. Shapiro, "Curve Formation Using Transforms for Pictures Governed by Differential Equations," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-8, No. 10, pp. 763-765, Oct. 1978.

[Shapiro79a] S.D. Shapiro and A. Iannino, "Geometric Constructions for Predicting Hough Transform Performance," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-1, No. 3, July 1979.

[Shapiro79b] S.D. Shapiro, "Use of the Hough Transform for Image Data Compression," *IEEE Conf. on Pattern Rec. and Image Processing*, Chicago, pp. 576-582, Aug. 1979.

[Sklansky78] J. Sklansky, "On the Hough Technique for Curve Detection," *IEEE Trans. on Computers*, Vol. C27, No. 10, pp. 923-926, Oct. 1978.

[Sloan80] K.R. Sloan, Jr., and D.H. Ballard, "Experience with the Generalized Hough Transform," *Proc. DARPA Image Understanding Wkshp.*, pp. 150-156, Apr. 1980. Also pub. as U. of Rochester Computer Science Department TR-83, October 1980.

[Sloan81] K.R. Sloan, Jr., "Dynamically Quantized Pyramids," *Proc. 7th Int. Jnt. Conf. on Artificial Intelligence (IJCAI-81)*, Vol. 2, pp. 734-736, 1981.

[Stockman77] G.C. Stockman and A.K. Agrawala, "Equivalence of Hough Curve Detection to Template Matching," *CACM*, Vol. 20, No. 11, pp. 820-822, Nov. 1977.

[Tanimoto77] S.L. Tanimoto and T. Pavlidis, "The Editing of Picture Segmentations Using Local Analysis of Graphs," *CACM*, Vol. 20, No. 4, pp. 223-229, Apr. 1977.

[Tisdale77] G.E. Tisdale, "A Digital Image Processor for Automatic Target Cueing, Navigation, and Change Detection," *Airborne Reconnaissance – Tactical/Real Time*, Reston, VA, Proc. SPIE, Vol. 101, pp. 112-9, April 1977.

[Tisdale79] G.E. Tisdale, A.R. Helland, and J. Shipley, *Automatic Target Cueing (AUTO-Q) System Engineering Model Design*, Westinghouse Systems Development Division, Baltimore, MD, Report 80-0488, September 1979, 60 pp.

[Touchberry77] R. Touchberry and R. Larson, "Symbolic Analysis of Images using Prototype Similarity," *Proc. Image Understanding Workshop*, pp. 78-82, Apr. 1977.

[Tsuji77] S. Tsuji and F. Matsumoto, "Detection of Elliptic and Linear Edges by Searching Two Parameter Spaces," *Proc. 5th Int. Jnt. Conf. on Artificial Intelligence*, Cambridge, MA, pp. 700-705, Aug. 1977.

[Tsuji78] S. Tsuji and F. Matsumoto, "Detection of Ellipses by a Modified Hough Transformation," *IEEE Trans. on Computers*, Vol. C-27, No. 8, pp. 777-781, Aug. 1978.

[van Veen81] T.M. van Veen and F.C.A. Groen, "Discretization Errors in the Hough Transform," *Pattern Recognition*, Vol. 14, pp. 137-145, 1981.

[WDESC78] *Intelligent Tracking Techniques*, Westinghouse Defense and Elec. Sys. Center, Systems Development Division, Baltimore, MD, Report 79-0503, December 1978, 75 pp.

[Wechsler77] H. Wechsler and J. Sklansky, "Finding the Rib Cage in Chest Radiographs," *Pattern Recognition*, Vol. 9, No. 2, pp. 21-30.

[Winkler78] G. Winkler and K. Vattrodt, "Measures for Conspicuousness," *Computer Graphics and Image Processing*, Vol. 8, pp. 355-368, 1978.

[Woolfson78] M.G. Woolfson, "Digital Area Correlation Tracker," *Proc. IEEE 1978 National Aerosp. and Electron. Conf. (NAECON 78)*, Dayton, OH, pp. 882-6, May 1978.

[Yam81] S. Yam and L.S. Davis, "Image Registration using Generalized Hough Transforms," *IEEE Conf. on Pattern Rec. and Image Processing*, Dallas, pp. 526-533, Aug. 1981.

[Zabele79] G.S. Zabele, *On Optimum Line Detection in Noisy Images using the Hough Transform*. Master's Thesis, Clarkson College, Jan. 1979.

Appendix B

The Phoenix Image Segmentation System: Description and Evaluation

THE PHOENIX IMAGE SEGMENTATION SYSTEM: DESCRIPTION AND EVALUATION

Technical Note

December 1982

By: Kenneth I. Laws, Computer Scientist

Artificial Intelligence Center
Computer Science and Technology Division

Program Development by:

Steven Shafer
Takeo Kanade
Duane Williams

Carnegie-Mellon University
Department of Computer Science

SRI Project 1009

The work reported herein was supported by the Defense Advanced Research Projects Agency under Contract No. MDA903-79-C-0588.

Foreword

The primary purpose of the Image Understanding (IU) Testbed is to provide a means for transferring technology from the DARPA-sponsored IU research program to DMA and to other organizations in the defense community.

The approach taken to achieve this purpose has two components:

(1) The establishment of a uniform environment as compatible as practical with the environments of research centers at universities participating in the IU research program. Thus, organizations obtaining copies of the Testbed can receive a continuing flow of new results derived from on-going research.

(2) The acquisition, integration, testing, and evaluation of selected scene analysis techniques that represent mature examples of generic areas of research activity. These contributions from participants in the IU research program will allow organizations with Testbed copies to begin the immediate exploration of applications of IU technology to problems in automated cartography and other areas of scene analysis.

The IU Testbed project was carried out under DARPA contract No. MDA903-79-C-0599. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the United States Government.

This report describes the PHOENIX segmentation package contributed by Carnegie-Mellon University and presents an evaluation of its characteristics and features.

Andrew J. Hanson
Testbed Coordinator
Artificial Intelligence Center
SRI International

Abstract

PHOENIX is a computer program for segmenting images into homogeneous closed regions. It uses histogram analysis, thresholding, and connected-components analysis to produce a partial segmentation, then resegments each region until various stopping criteria are satisfied. Its major contributions over other recursive segmenters are a sophisticated control interface, optional use of more than one histogram-dependent intensity threshold during tentative segmentation of each region, and spatial analysis of resulting subregions as a form of "look-ahead" for choosing between promising spectral features at each step.

PHOENIX was contributed to the DARPA Image Understanding Testbed at SRI by Carnegie-Mellon University. This report summarizes applications for which PHOENIX is suited, the history and nature of the algorithm, details of the Testbed implementation, the manner in which PHOENIX is invoked and controlled, the type of results that can be expected, and suggestions for further development. Baseline parameter sets are given for producing reasonable segmentations of typical imagery.

Table of Contents

Section 1

Introduction

PHOENIX is a program for segmenting an image into homogeneous regions. It combines histogram analysis with spatial analysis to find connected regions having uniform color or other properties. Small noise patches are merged with their surrounding or neighboring regions. Regions may then be further segmented by the same algorithm.

Many researchers have contributed to this segmentation technique, as documented in Section 3. The current PHOENIX program was designed by Steven Shafer and Takeo Kanade at Carnegie-Mellon University (CMU), with much of the programming done by Duane Williams and Marc Lowe. Drs. Raj Reddy at CMU and Hans-Hellmut Nagel at the University of Hamburg have guided and supervised much of the development.

The CMU PHOENIX code has been adapted for the DARPA Image Understanding Testbed at SRI International. Many of the testbed support routines provided by CMU were adapted for the Testbed by Kenneth Laws at SRI. Particular credit is due to Steven Shafer for the CI driver and related string manipulation routines, David Smith for the image access software, and David McKeown, assisted by Steve Clark, Joe Mattis, and Jerry Denlinger, for the Grinnell display software. All of this software is written in the C language.

Very few changes were required in the PHOENIX software or in the algorithm itself. The information in this document should thus be considered supplementary to the material cited in the references. User documentation provided by CMU [Smith80, Clark81, McKeown81, Shafer82] forms the basis for some sections of this report.

This document includes both a users' guide to the PHOENIX segmenter and an evaluation of the algorithm. The initial portion introduces the segmenter and describes it in general terms. Section 2 briefly describes the algorithm and the tasks for which it is appropriate; Section 3 surveys the historical development of these techniques and presents the current algorithm in detail.

The next portion of this report constitutes a users' guide. Section 4 describes the current Testbed implementation and how it differs from the original CMU contribution. Section 5 instructs the user in the mechanics of using the PHOENIX software.

The remainder of the report body summarizes the evaluation results. Section 6 describes in detail the meaning of the user-specified parameters, documents the performance that may be expected in various circumstances, and presents the results of evaluation tests. The groups of parameter values developed in this section are a significant scientific contribution. Section 7 outlines a number of suggestions for improving the algorithm and its implementation. Section 8 presents conclusions, including a brief statement of the special strengths and weaknesses of the PHOENIX approach.

Appendix A suggests alternate approaches to similar data analysis problems, and Appendix B gives the details of the connected-component extraction algorithm. An

extensive reference list provides entry points to the image segmentation literature cited in the text.

Section 2

Background

This section presents a management view of the PHOENIX program. The segmentation algorithm is briefly sketched. Typical applications and potential applications requiring further development of the algorithm are discussed, and related applications for which other algorithms are better suited are noted.

2.1. General Description

PHOENIX is a program for segmenting images into homogeneous connected regions. An input image typically has red, green, and blue image planes, although monochrome images, gradient and texture planes, and other pixel-oriented data may also be used. Each of the data planes is called a *feature* or *feature plane*.

Figure 1.1 illustrates the image segmentation process. Segmentation begins with the entire image considered to be a single region. Phoenix "fetches" this region and attempts to segment it. If it fails, the program halts and waits for further instructions; if it succeeds, it fetches each of the new regions in turn and attempts to segment it. A *segmentation queue* keeps track of the regions that are awaiting further analysis; a *terminal queue* keeps track of those that have been declared *terminal* regions.

Having fetched a region, PHOENIX computes a vector of intensity counts (a *histogram*) for each feature plane. Thresholds (or histogram *cutpoints*) are selected that are likely to isolate significant homogeneous regions in the image. A set of thresholds for one feature is called an *interval set* because each threshold defines a histogram interval extending from the previous cutpoint to and including the new one.

The most promising interval sets are passed to a spatial analysis phase that thresholds the corresponding feature plane and extracts connected components. Very small connected patches are considered noise and are merged with surrounding regions.

The feature and interval sets providing the best segmentation (*i.e.*, the one with the least noise area) are chosen. Each of the resulting segments is added to the knowledge base and segmentation map and is queued for further segmentation using the same algorithm.

This process halts when the recursive segmentation reaches a preset depth, when all regions have been segmented as finely as various user-specified parameters permit, or when the user terminates execution. The segmentation is saved, and may be reloaded and edited or continued later. The resulting region map and region description file may be used by other programs.

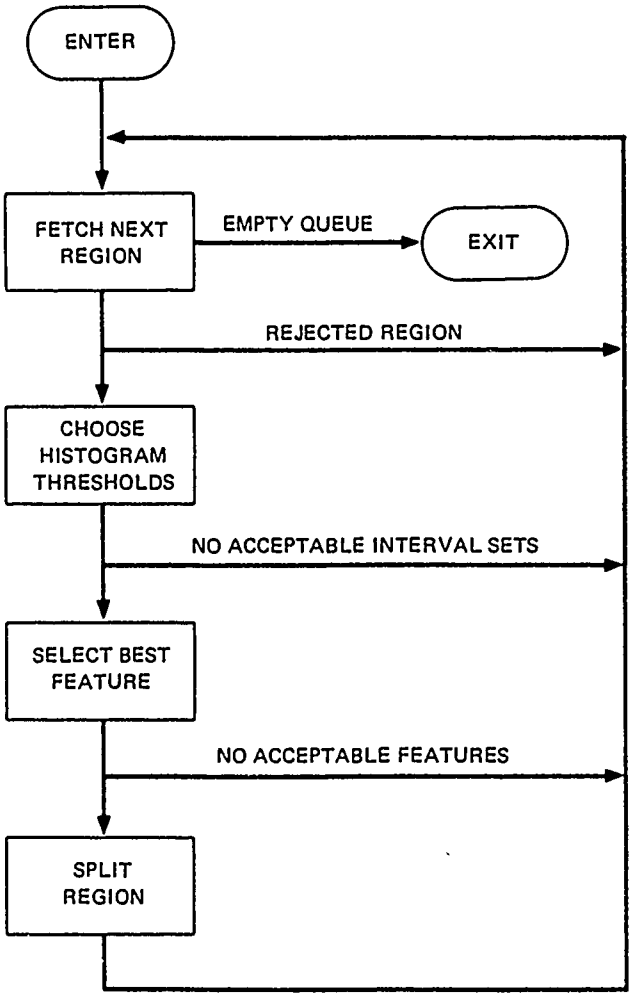

FIGURE 1.1 BASIC CONTROL SEQUENCE

2.2. Typical Applications

The PHOENIX program may be used in any application requiring that an image be partitioned into homogeneous regions. This segmentation may be useful in itself, or may be a precursor to a semantic partitioning that assigns meaningful labels to composite regions.

The initial segmentation by itself is most useful for image coding applications. Since there are far fewer regions than pixels, it may be efficient to store or transmit an image as a list of regions. This would be particularly effective in time-sequenced imagery, since only those regions that change need to be coded for each frame. The amount of compression possible depends on scene content and on the acceptable coding error. One scheme [Yan77] uses run coding to transmit the region map, or *cartoon*, and then adds a low-amplitude correction signal to fill in the details.

This same separation of the image signal may be useful in image enhancement. Enhancement within each region separately can bring out details that are otherwise obscured by illumination effects. This is similar to separate processing of low-frequency and high-frequency signal bands, but preserves edge structure better.

Region boundaries located by PHOENIX may be used to measure image blur or the transfer function of the imaging system. This information can be used in image restoration and in estimating scene depth from the amount of blur.

The PHOENIX region descriptions may be used for microscopic particle counting or for counting of nonoccluded industrial parts. PHOENIX will not distinguish touching objects, but area measurement (for uniform particles) or shape analysis (*e.g.*, [Arcelli71, Brenner77, Lemkin79, Jain80, Rutkowski81]) can make this separation. Simple size and shape descriptors may also be adequate for some medical cell classification problems.

Another application is in macrotexture analysis. Macrotextures are those that have large primitive elements forming some type of pattern. A checkerboard is a regular macrotexture; orchards, agricultural fields, and housing developments in aerial images are less regular; and tree leaves or microscopic mineral domains may be very irregular. The first step in analyzing such a texture is to identify the primitive elements, either by template matching or by segmentation [Tomita82].

Segmentation maps may also be useful in registration (*i.e.*, alignment) of two images [Ratkovic79a-c]. The two maps are first matched, giving an approximate global registration. The low-amplitude correction signals for each pair of regions are then used for precise local registration. This seems to be a good way to determine image warp coefficients, and may also be useful in tracking slowly moving objects in cluttered backgrounds.

An attempt has been made [Price76, Price78a, Price78b] to use region information for change detection in complex urban and industrial scenes. Many regions remain constant from one image to another, but others might move or change form. Region descriptions in either image that could not be matched (in shape, position, and possibly intensity) were specially flagged for user attention. The method was sophisticated enough to match similar regions at differing positions, but could not determine whether they were two similar objects or a single one that had moved.

Segmentation's most promising application, although one where it has yet to prove its worth, is in general-purpose image understanding [Fischler79, Faugeras80,

Rubin80, Ohta80b]. Segmentation and linear delineation are considered to be the first steps in feature extraction, followed by texture analysis, determination of surface orientation, and object recognition. These research topics will be discussed below.

2.3. Potential Extensions

The following applications might be feasible if PHOENIX were modified, used in a non-standard fashion, or integrated into a more sophisticated system.

Crude region knowledge may be the key to obtaining more precise knowledge: this is known as planning. Preliminary segmentation (often on a reduced image) can be used to determine which areas should be examined in more detail. Specialized structure detectors may then be applied within the regions or along the region boundaries. If the analysis is done in real time, higher resolution data may be obtained by rescanning portions of the original scene. In missile guidance, for instance, higher resolution imagery becomes available as the missile approaches its target.

Many natural scenes are better described by textured regions than by regions of homogeneous intensity. PHOENIX can be used to find textured regions if texture feature planes are provided as input. Many texture measures or transforms have been suggested [Haralick73, Carlton77, Schachter77, Mitchell78, Tanimoto78, Coleman79, Schachter79, Laws80, Lee82], but their use in PHOENIX will probably require more sophisticated feature selection and processing.

If texture-based segments are available, it becomes feasible to classify each region as to its texture type or materials category (assuming sufficient resolution). Adjacent regions that receive the same classification may then be merged to produce a better segmentation. (Note, however, that it may or may not be desirable to merge two fields that have the same crop type but different plowing directions, or two cloud patches that may be at different elevations. The merging algorithm needs knowledge about both the scene domain and the intended application.)

Segmer'. maps may also be used as input to an object identification or intelligent cuein~ ystem. The system should be capable of recognizing objects composed of several regions. In some circumstances it may also have to guess at those which are contained within part of a region and, if possible, use additional processing to confirm the hypothesis.

2.4. Related Applications

This section describes applications that are similar to PHOENIX segmentation applications, but differ in some fundamental fashion. While the difficulties with applying PHOENIX might be overcome, other techniques would often be more appropriate.

Cueing is the initial detection of interesting objects in a scene. While cueing using a segmentation map may be possible, the effort of computing the map may be far greater than that required for threshold detection, interest-point or corner detection, unusual-pattern detection [Haralick75b, Winkler78], statistical classification, blob detection [Klein77, Deal79, Tisdale79, Danker81], prototype matching [Aggarwal78], or other techniques. Thus PHOENIX should only be used for cueing if the segmentation is required for other purposes.

Background

Object recognition is often combined with cueing when only certain objects are of interest. The problem of locating predictable signatures is best solved with matched filtering or template matching. A particularly efficient and flexible template matching method is based on the Rochester generalized Hough transform. (For a review see [Laws83].) More general object detection requires image understanding, and segmentation may be a useful preprocessing technique.

Linear delineation is the extraction of image edges, region boundaries, and elongated features. Region boundaries can be found using PHOENIX, but thin, elongated, or nonclosed features tend to be missed. A complete image understanding system will need both region extraction and linear delineation operators [Nevatia77a]. Representative techniques are described in Appendix A.

PHOENIX segments images using a recursive thresholding algorithm. The regions identified at each step are relatively uniform in one feature, and terminal regions tend to be uniform in all features. In some domains this method will fail. In extracting an illuminated sphere or cylinder, for instance, the important property is continuity rather than uniformity. Edge-based linear delineation systems are much better at segmenting smoothly-varying imagery.

Image understanding and object recognition require that many sources of knowledge be applied [Barrow75]. In particular, the system may require knowledge of sensor characteristics [Garvey76a], 3-D or physical domain knowledge [Fischler79, Fischler82], illumination and reflectance models [Horn77], semantic knowledge of likely adjacencies [Yakimovsky73a, Yakimovsky73b, Feldman74, Barrow76, Tenenbaum76a, Tenenbaum76b, Tenenbaum80], or models of likely target configurations [Price81]. It is not yet known whether segmentation should be a precursor to such analysis or should be tightly integrated with it.

Section 3

Description

This section presents the history of recursive image segmentation and a detailed statement of the PHOENIX algorithm. The historical information is intended to clarify the major issues in recursive segmentation and to provide entry points into the literature.

3.1. Historical Development

Histogram thresholding was an early segmentation technique [Prewitt66]. One or more histogram cutpoints were chosen near valleys in the intensity histogram. Connected-components analysis was then used to extract regions entirely darker or brighter than the corresponding intensity threshold level. There were difficulties, however; if an image contained many regions with overlapping histogram peaks, there would then be no obvious or useful thresholds. One solution, used by Chow and Kaneko [Chow70], was to partition an image into smaller subimages until distinct peaks appeared or the windows became so small that the histograms degenerated.

The earliest use of recursive region-splitting by histogram thresholding was for analysis of black-and-white cell images [Prewitt70]. Connected components were extracted from the thresholded image and were used for further segmentation. For other early approaches to segmentation see Appendix A.

Tsuji and Tomita [Tsuji73, Tomita73] at Osaka University used recursive region-splitting to segment macrotexture images. The shape statistics of the primitive elements were compiled into histograms. The smoothed histogram with the most distinct valleys was used for classifying the elements into two or more sets. Connected components were extracted (with some overlap allowed), and very small regions were merged with their neighbors, if possible. Boundaries of the regions were computed and compared with scene models, and those regions not corresponding to known object types were scheduled for further partitioning.

Robertson *et al.* [Robertson73] at Purdue University pursued the notion of histogram thresholding for segmentation of multispectral scenes. They also used recursive segmentation along rectangular boundaries, foreshadowing later development of the quadtree segmentation representation.

Several researchers investigated segmentation of natural textures where primitive elements could not be extracted. Kasvand [Kasvand74] used a primitive constant threshold with texture measures based on local standard deviation, gradient, second derivative, and other features. Zucker *et al.* [Zucker75] computed response to a spot detector at every point in a scene and tried to segment the resulting histogram. Satisfactory results could only be obtained if the spot detector was approximately matched to the texture coarseness and if nonmaximal suppression was used to reduce blurring due to the measurement window size. These researchers did not use recursive segmentation.

Other researchers were attempting to segment color imagery using multidimensional histogram analysis. Tenenbaum *et al.* [Tenenbaum74] at Stanford Research Institute (now SRI) projected the three-dimensional color space onto the chromaticity plane and segmented on pixel hue. Even with thresholds based on prominent scene objects [Tenenbaum75], there were difficulties with overlapping hue distributions in landscape scenes and with color-coordinated decor in indoor scenes, as well as with an abundance of small texture regions. Neither texture features nor recursive segmentation were used.

Ohlander [Ohlander75] at Carnegie-Mellon University adapted the Tsuji algorithm for color images by computing histograms of three color features (RGB) and six color transformations (YIQ and HSD). A simple texture feature was also computed to identify microtexture regions. These features were used for recursive segmentation within arbitrary region boundaries. At each stage the histogram with the most prominent isolated peak was chosen for segmentation. Pixels related to the peak were then extracted and represented by a bit mask. (All those with higher or lower feature values were represented by the complement of the mask over the original region.) High-resolution, and hence large pictures and long processing times, was needed to accurately isolate textured regions and locate objects in natural imagery. Interactive thresholding inside textured areas was also necessary to segment a city skyline scene.

Schachter *et al.* [Schachter75] at the University of Maryland were also studying color image segmentation at this time. They chose to store the full three-dimensional histogram as a binary tree. They report that a leaf node is needed for every five or ten pixels in the image. (This would increase if texture measures were included.) Clusters in the tree were found by a single-linkage (or chained nearest-neighbor) algorithm. Nonrecursive segmentation was then done by assignment of pixels to the cluster classes. A similar method was later used for texture segmentation of monochrome imagery [Schachter77].

Kender at CMU analyzed the color transformations used by Tenenbaum and Ohlander; he concluded that inherent singularities and quantization effects were capable of introducing false peaks and valleys [Kender76, Kender77]. This effect is particularly noticeable in the hue feature, but also affects saturation and other normalized chromaticity coordinates. He recommended that saturation only be used in regions of high luminance, with hue used only in high saturation as well. (Note that most natural imagery has low to moderate saturation.) The YIQ transform used in color television transmission was found to have fewer problems, although its usefulness in segmentation was not evaluated. Kender also proposed an improved computational algorithm for hue.

Mui *et al.* [Mui76] brought together iterative segmentation and spatial analysis for the segmentation of blood cell images. An initial threshold segmentation was used to determine scene parameters and initial histogram cluster centers. Refined clusters were then found in the "color-density" histogram, and these were mapped back to the spatial domain. Similar techniques have been used in many medical image-analysis systems [Aggarwal77, Cahn77].

A key concept of later segmentation systems is *planning*, or heuristic guidance. Planning was introduced by Kelly [Kelly70] in the recognition of human images. A reduced image was first used to find the face or body outline, then individual features were sought in higher-resolution imagery. *Ad hoc* rules were used to identify the mouth, eyes, pupils, and other facial features. Kelly later applied planning to edge detection [Kelly71]. Planning, or hierarchical image feature extraction, was also the

foundation of other pyramid or processing-cone systems [Uhr72, Harlow73, Hanson74, Tanimoto75, Klinger76, Levine76, Dyer81].

Price at CMU brought together recursive region-splitting and planning [Price76]. His PLAN program for segmentation and symbolic matching used a refinement of the Ohlander algorithm on a reduced image, then applied the same thresholds within a slightly enlarged mask area in the full-resolution image. This two-stage approach reduced segmentation time by a factor of about ten. The color features used were a modification of Ohlander and Kender's YIQ and HSD, although LANDSAT spectral band features were also used. Price introduced several texture measures for monochromatic segmentation and added a spatial smoothing step to remove small holes from the binary masks. Less human interaction was required during histogram analysis, region extraction, and database maintenance than for Ohlander's system.

Aggarwal *et al.* [Underwood77, Ali79, Sarabi81] at the University of Texas have used a different approach for the segmentation of color images. They have mapped the image data into a three-dimensional intensity and chromaticity histogram. The bivariate marginal histograms may be displayed for interactive cluster identification, or a binary tree structure similar to that of Schachter *et al.* may be used for automated cluster identification. A version of the system used discriminant analysis to detect diseased citrus trees in infrared color imagery. An advantage of the chromaticity coordinates is that shadow regions in the image may often be easily identified.

Ohta *et al.* have further investigated color transforms for recursive segmentation [Ohta80a, Ohta80b]. They computed color histograms using the Karhunen-Loeve color transform – an expensive method because the transform is different for each region. Ohta found that the transform principal axes tended to cluster around

$$I_1 = red + blue + green$$

$$I_2 = red - blue$$

$$I_3 = 2\,red - (green + blue)$$

and recommended that these features be used. (The second and third features may be negative, so that either an offset is necessary or the segmentation code must be able to handle negative pixel values.)

Ohta's transform is similar to the YIQ system and to the opponent color process recommended by several authors [Sloan75, Nagin78]. The transform is linear, and hence avoids the instabilities that Kender found in saturation, hue, and normalized chromaticity coordinates. Nagin expressed some theoretical reservations about his own opponent features, but concluded that they "consistently provided more discrimination than the original RGB data."

Nagin also explored the use of "conservative" histogram thresholding (*i.e.*, suppressing doubtful classifications) combined with region growing, and showed how the image segmentation algorithm itself could be used for segmentation of two-dimensional histograms [Nagin77, Nagin78]. Other two-dimensional histogram analysis systems have been built by Milgram *et al.* [Milgram79, Milgram80] to segment monochrome images using pixel edge-strength in addition to intensity.

Meanwhile, work on recursive segmentation has continued at CMU. The current PHOENIX program is a VAX 11/780 implementation of Shafer and Kanade's KIWI program for the PDP 11/40. A related system named MOOSE [Shafer80] is being studied

at the University of Hamburg for symbolic motion analysis. The algorithm used in these systems is described below. The use of multiple histogram intervals, spatial analysis look-ahead, and the interactive control system are major innovations incorporated into PHOENIX.

3.2. Algorithm Description

Image segmentation reduces a pixel array to a map or list of significant regions. This greatly reduces the number of entities to be dealt with while increasing our knowledge about the image. (The increased knowledge, or information, may be measured by the reduced number of bits required to code the image. More importantly, the extracted segments are usually related to objects in the imaged scene.)

It is difficult to talk about the complexity of the segmentation task without discussing particular techniques, although this has been attempted [Gurari82]. For a survey of statistical image models for classification and segmentation see [Rosenfeld79].

There are many approaches to image segmentation, and each has its domain of applicability. Edge-based methods attempt to derive closed regions from linear discontinuities. Region-growing methods extend small homogeneous regions by incorporating neighboring pixels or regions. Region-splitting (or thresholding) methods subdivide initial regions by identifying more homogeneous subregions. All of these techniques are discussed further in Appendix A. The following describes the PHOENIX algorithm for image segmentation.

3.2.1. General Approach

The PHOENIX algorithm is a region-splitting technique. It has the advantage that a partial segmentation is meaningful, and only those regions satisfying higher-level criteria need to be considered for further segmentation.

A scene is assumed to be composed of numerous connected regions, each of which is approximately uniform in texture and, if untextured, in all of its other pixel properties. The luminance image of an untextured scene then resembles a mosaic of flat-topped "mesas." These regions may be related to portions of objects, to whole objects, or to clumps of objects. (We will temporarily ignore shadows, occlusions, and other complications.)

The segmentation algorithm must identify image regions that correspond to such scene regions. The job is complicated by imaging blur, spatial and intensity quantization, and other artifacts of the imaging process. The most serious problems, however, arise when the scene contains sloped facets [Haralick80] or continuous gradients that violate the assumed mesa model.

PHOENIX finds uniform regions by recursively splitting nonuniform ones, beginning with the whole image, into smaller regions. (See Appendix A for a discussion of splitting techniques.) The connected components associated with each intensity slice are then extracted. This process is not necessarily cheap, but there is evidence that it is well-suited to a parallel architecture such as the human visual system. Price [Price76] lists counts of machine operations required to perform many of the recursive segmentation steps. The amount of computation per region is nearly independent of the number of subregions found, so there is a bonus if the technique finds many subregions in a single pass.

One way to locate many regions is to analyze the histograms of many features. PHOENIX and the Ohlander-type segmenters typically use three independent color features per pixel, plus one or more texture features. (For monochrome imagery, only intensity and the texture features are available.) Although joint histogram analysis is possible, it is of the same order of difficulty as the original image segmentation problem. PHOENIX opts for simplicity by analyzing only the one-dimensional marginal (or single-feature) histograms, augmented by one-dimensional histograms of linear or nonlinear feature combinations.

Each pass of the Ohlander/Price algorithm found segments related to a single peak in a single marginal histogram. The PHOENIX program is able to use multiple histogram intervals to increase the number of regions found in one pass, although typical operation uses only one threshold per feature in order to minimize noise and segmentation errors.

3.2.2. Color Features

Although color transformations are not strictly a part of the PHOENIX program, they are fundamental to its theoretical basis and to its typical operation.

Color features are needed when two regions to be distinguished have similar intensity (and texture), but different hue or saturation. Even if the regions are not adjacent, their intensity histograms will overlap and prevent discrimination. Hue, saturation, or other color features may be used to break the deadlock.

Color features for image processing research are typically generated by scanning a color photograph through color filters (*e.g.*, Wratten filters 25, 47B, and 58) to get red, green, and blue feature planes. Real-time systems often use an electronic color camera to generate YIQ features, which correspond roughly to perceptual brightness, cyan *vs.* orange, and magenta *vs.* green. ('I' stands for *in-phase*, 'Q' for *quadrature*.) The two color systems are equivalent, and we shall henceforth assume that the primary input is in the RGB coordinates.

Each color system constitutes a three-dimensional color space, that can express most of the colors perceived by humans. (The full detailed spectrum that, *e.g.*, astronomers and physicists depend upon has been lost, just as it is in the human visual system.) A few purples and highly saturated colors are not precisely representable, the colors recorded with different films or cameras may differ, and digital quantization limits the fineness of color distinctions, but the three-component representation is adequate for most purposes.

Typical quantization is eight bits per color axis, or 16.8 million cells for an entire three-dimensional histogram. Repeated cluster analysis in such a histogram is not attractive, although nonhistogram methods of multidimensional pattern recognition are available. The PHOENIX package instead uses an adaptation of the one-dimensional histogram segmentation developed by Tsuji, Tomita, and Ohlander.

Any one-dimensional histogram is equivalent to a projection of the three-dimensional data onto a line (or curve) through the color space. If the scene contains many regions, their histogram peaks are likely to overlap and obscure any useful details in the composite histogram. The overlap is different for projections at different angles, and it is often possible to isolate peaks from some of the regions by using many different projections.

Several projections, or transformations, were discussed in Section 3.1; many others are possible. The authors of PHOENIX have generally stayed with Ohlander's choice of RGB, YIQ, and HSD (hue, saturation, and intensity) projections, although they note the instabilities of the HSD system near the D axis [Shafer82]. (The HSD system is also known as the HSI or IHS system. The symbol D is used here to avoid confusion with the YIQ system. It comes from *density*, a measure of the amount of silver deposited at a given point in a photographic negative.)

The color transforms are generally computed by the method of Kender [Kender76, Kender77]. The YIQ coefficients are

$$Y = 0.509R + 1.000G + 0.194B$$

$$I = 1.000R - 0.460G - 0.540B + M$$

$$Q = 0.403R - 1.000G + 0.597B + M$$

where M is the highest possible intensity value in the original RGB features, typically 255. These formulas have been linearly scaled to maintain quantization accuracy (via the unit coefficient). The addition of M is simply for convenience in digital representation. (The Q feature can be negated before adding M to better match the green gun on a color monitor.)

The HSD coordinates were introduced by Tenenbaum *et al.* [Tenenbaum74] to mimic human color perception. Briefly they are

$$H = \arccos \frac{(R-G)+(R-B)}{2\sqrt{(R-G)(R-G)+(R-B)(G-B)}}$$

$$S = m\left(1 - 3\frac{\min(R,G,B)}{R+G+B}\right)$$

$$D = \frac{(R+G+B)}{3}$$

where m is the maximum desired saturation value. Hue is normalized by subtracting it from 2π if $B>G$, and some care must be taken in rounding the values near 2π if the number is quantized. Note that these formulas contain singularities due to division by zero: Kender recommends detecting these cases and treating them as special values. See [Kender76, p. 35] for a computational algorithm.

3.2.3. Texture Features

Only the intensity feature (D or perhaps Y) is available for monochrome imagery. This is occasionally adequate for segmenting simple scenes with large objects (as in cell counting [Prewitt70] or some types of industrial inspection), but aerial scenes usually show so many regions that the composite histogram is unimodal. Recursive segmentation can only proceed by using additional texture features or special control strategies (see Section 3.2.6).

Structural texture features can be used [Tsuji73, Tomita82], but the PHOENIX program is best adapted for statistical texture features that can be measured at each point. There are many such measures. Ohlander used a simple Sobel-edge "busyness" feature to identify textured regions in color imagery. Price used local edge density, variance, and range to segment aerial and side-looking radar imagery. (He suggested that local minimum or maximum pixel values could be used to distinguish some regions.) Fourier and other spatial transforms are popular [Pavlidis75,

Tanimoto78]. Local gradient or edge strength could also be used, although the histogram analysis must be more sophisticated [Milgram79, Milgram80].

Ohlander used texture only to remove busy regions from further consideration by the color segmentation system. Price also used texture this way, but was able to segment monochrome images using texture features in place of color features. PHOENIX carries this integration even further by using a limited form of look-ahead: at each step only those features producing "clean" spatial segmentations are kept. Thus texture and color features may be used together. (A more intelligent system would understand the nature of each feature plane, and failure of a color feature to provide compact regions would activate a texture analysis subsystem. This has not yet been tried.)

3.2.4. Histogram Analysis

It was stated earlier that each region in a scene is modeled as a uniform patch in the image. Such a model implies that the histograms should contain only sharp spikes. A more appropriate model, allowing for some texture and imaging effects, is that each region produces a noisy Gaussian peak in the histogram.

Methods do exist for decomposing a function into Gaussian peaks. This is known as the mixture density problem [Wolfe70] and is important in information theory, statistics, chemistry, and other fields. Very little of this theory has been applied to image processing [Chow70, Rosenfeld76b, Postaire81]. PHOENIX is able to use its spatial knowledge to avoid the difficulties of these methods, although at the cost of making some errors in threshold placement. These errors cause the break-up of some small regions and shifting of region boundaries on others.

Ohlander and Price used a hierarchy of heuristic rules for selecting the most prominent peak within a set of histograms [Ohlander78, Price79, Nevatia82]. The peak was delimited by two thresholds that defined an intensity interval and its complement. PHOENIX uses similar heuristics, but concentrates on the valleys (*i.e.*, local minima) in the histogram set. Usually a single valley, resulting in one threshold and two intervals, is selected for each feature. Spatial analysis is then used to select the best threshold/feature combination. Using only one threshold per pass reduces the chance of segmentation errors, although it does increase the number of passes required.

The PHOENIX histogram analysis uses region growing instead of recursive segmentation. A histogram is first smoothed with an unweighted moving average. It is then broken into intervals such that each begins just to the right of a valley (*i.e.*, at the next higher intensity), contains a peak, and ends on the next valley. A valley is considered to be the right *shoulder* of its left interval and the left shoulder of its right interval. The leftmost and rightmost intervals always have exterior shoulders of zero height.

A series of heuristics is then applied to screen out noise peaks. Each test is applied to all the intervals in the histogram (providing there are enough intervals for the test to be meaningful – two for some tests, three for others). When an interval is eliminated, it is merged with the neighbor sharing the higher of its two shoulders. The screening test is then applied again to the merged interval; previous tests are not reapplied.

Peak-to-shoulder ratio is tested first. An interval is only retained if the ratio of

peak height to the higher of its two shoulders, expressed in percent, is at least as great as the **maxmin** threshold. (See Section 5.1 for more about this and other user-supplied thresholds.)

Peak area is then compared to an absolute threshold, **absarea**, and to the **relarea** percentage of the total histogram (or region) area. Only peaks larger than these thresholds are retained.

The intervals surviving to this point should be reasonable candidates, and it is reasonably safe to use global histogram descriptors in the test conditions. The second-highest peak is now found, and peaks less than a percentage, **height**, of it are merged. The lowest (interior) valley is then found, and any interval whose right shoulder is more than **absmin** times this is merged with its right neighbor. (The parameter seems to be misnamed since the criterion is relative rather than absolute.)

A final screening is made to reduce the interval set to **intsmax** intervals. This is done by repeatedly merging regions with low peak-to-shoulder ratios until only **intsmax**−1 valleys remain.

A score is also computed for each interval set. This score is the maximum (apparently MOOSE used the minimum) over all intervals of the function

$$1000\ \frac{\textit{peak height} - \textit{higher shoulder}}{\textit{peak height}}$$

This formula assigns the maximum score to an interval set containing a peak with shoulders of zero height.

3.2.5. Spatial Analysis

PHOENIX next chooses features (and corresponding interval sets) for spatial evaluation. The best **isetsmax** interval sets will be chosen, provided that each has a score of at least **absscore** and at least **relscore** percent of the highest interval set score.

Each selected interval set is then tested for segmentation quality. The histogram cutpoints are applied to the feature plane as intensity thresholds and connected components are extracted. (See Appendix B for the extraction algorithm.) Applying the thresholds introduces *segmentation noise* of three kinds: border placement errors, small *noise patches* that do not correspond to scene objects, and thin *necks* connecting patches that should be separated.

Border placement errors occur when the threshold separating two patches is influenced by histogram contributions from other nonadjacent patches. The effect can be so severe that small regions are split apart and/or absorbed into neighboring regions. This can be combated by conservative thresholding [Milgram79] or by some type of post-analysis using the statistics of only the two regions involved. (See [Milgram77] and [Milgram80] for methods of combining edge evidence with histogram analysis.) PHOENIX currently ignores such errors.

Price's PLAN program used a fast (but still time-consuming) spatial smoothing step to eliminate noise regions and connecting necks. Unfortunately the method also rounded corners and straightened thin diagonal objects. A more intelligent method would need to determine which pixels were noise regions or necks and to alter only those.

The PHOENIX spatial analysis is able to deal with noise regions, but not with connecting necks. (Large regions joined by a neck will usually be split in a later segmentation pass.) PHOENIX calls a subroutine to determine whether a connected patch is a noise region or a true region. At present this subroutine performs only an area test, with patches smaller than **noise** pixels considered to be noise regions.

After each feature has been evaluated, the one producing the least total noise area is accepted as the segmentation feature (providing that the noise area is less than a percentage, **retain**, of the total region area). The subregions obtained with that interval set are added to the segmentation record and the next segmentation pass is scheduled.

3.2.6. Control Strategies

PHOENIX uses a spatial analysis look-ahead to improve the selection of a segmentation interval set, just as modern chess-playing programs use dynamic evaluation to validate moves that seem good to a static evaluator. Spatial analysis improves on selection by the interval set score about 40% of the time [Shafer82], although the order in which features are selected may have little effect in many of these cases.

Several other high-level control strategies have been proposed to overcome specific problems. Ohlander and Price, for instance, used ordering of texture and color feature sets to guarantee that some features would be tried before others. PHOENIX has no such ordering because the spatial analysis rejects any inappropriate feature that would cause the breakup of a region. The program developers recognize, however, that such methods might save computation time or be otherwise useful; they have added such a facility to a later version of PHOENIX than is documented here [Shafer82].

Two methods of reducing computation time are *planning* and *focusing*. Planning was discussed in Section 3.1. It involves use of thresholds and region masks derived from reduced images to speed segmentation of full-resolution images. PHOENIX does not incorporate planning.

Focusing is the use of interest operators, motion detectors, or higher-level knowledge to crop the image around objects of interest [Shafer80]. This concentrates expensive resources on appropriate tasks, but does run the risk of missing unexpected objects in the scene. PHOENIX does not include an automatic focusing mechanism, but the user may control which regions of the image are to be segmented further. The user may also "prune" regions where the subregion structure turns out to be uninteresting.

Another difficult problem is the initiation of action when the original set of features is insufficient to identify a usable threshold. This often occurs in monochrome segmentation, because the single luminance feature has insufficient degrees of freedom for separating the overlapping peaks of many small regions. Texture features also tend to be unimodal unless the scene contains large areas of distinctive texture (such as agricultural fields [Keng77a]).

Color features are typically multimodal, making it easy to begin segmentation of even large scenes. Some possible explanations for this phenomenon are:

- Co-evolution of natural visual systems and of the environment they operate in may have produced a teleological segmentation of natural scenes into colored areas corresponding to functional entities. (Note that color is nearly always absent in caves and in deep ocean environments.) Man has continued this trend in the construction and decoration of technological artifacts. While colored objects are "intended" to contrast with their backgrounds, natural textures are more often accidental or intended for concealment. Further, since our understanding of texture is poorly developed, the texture measures we are using may have little discriminating power to begin with.
- Color is a point property, and can be measured very precisely. Texture is a local neighborhood property, and current methods of computation inherently blur the scene. If texture is measured over 15x15 windows, a single pixel from a different texture source contaminates the measured texture at 224 locations around it. The measurement windows of any two adjacent pixels have 93% overlap. This tends to smooth the texture histograms. For methods to combat this (by nonmaximal suppression and by choice of window size) see [Zucker75].
- Perceptual color is a three-dimensional space. Projecting it to a one-dimensional space (*e.g.*, luminance) often destroys cluster separability; multiple projections must be used to retain sufficient degrees of freedom. Texture space may well have dozens of dimensions, and we have been measuring it along too few axes for good separability.
- Color features are measured through "leaky" filters that permit some response to other colors. Consider a picture of a red flower against a green background. If the color filters were ideal, the red histogram would have a single peak representing the flower and the green histogram would have a single peak due to the background. Only by blending the two histograms, as occurs now with our broad filters, could histogram analysis find a starting point. Many of our texture measures are designed to be orthogonal, and it may similarly be necessary to use linear and nonlinear combinations of texture features to augment their effectiveness. (Combinations of texture and color may also be possible [Rosenfeld80].)
- Grahame Smith of SRI has suggested that multiple filtering may also play a role. Imagery for image understanding research has typically passed through at least two filtering processes during capture on film and subsequent digitization. The combined effect may introduce deeper histogram valleys than were present in the original scene. Texture measures are not subject to these influences.
- As Kender has pointed out, quantization and aliasing in digital transformations introduce false peaks and valleys into an otherwise uniform histogram. The effect on natural scenes has not been fully studied, but it is very likely that hue and perhaps saturation exhibit these effects. Various noise sources, particularly the picket fence effect of digital contrast improvement, may also introduce sharp peaks and valleys into the color histograms. Texture measures are often computed using floating-point arithmetic, and so avoid these effects.

Whatever the reason, luminance and texture features alone are often too unimodal to initiate segmentation of a large region. A higher-level control strategy is needed to get the segmenter off dead center. Once it has broken the image into regions, there is often enough peak separation to continue to a reasonable segmentation.

Price solved this problem using *partitioning*. The image was arbitrarily broken into s.. iler sections, and the histogram of each was computed. Each histogram was treated as the histogram of a feature over the whole image. Thus if a peak was found in the histogram of one image section, it was used to threshold the entire image. PHOENIX has no such mechanism, although its spatial analysis step would make such an action less dangerous.

Another, much simpler, heuristic would be to gradually weaken all thresholds until some histogram became segmentable. In the limit this would require that a feature threshold be arbitrarily chosen. Although this sounds crude, it may be exactly what is currently happening in the color domain. If the arbitrary threshold proved effective, any inappropriate segmentation that it caused could later be undone in an editing step.

Section 4

Implementation

This section documents the SRI Testbed implementation of PHOENIX, which is very little changed from the original CMU implementation. It is intended as a guide for system maintainers and for programmers making modifications to the PHOENIX system. The terms used in this section may be a little cryptic: they are either defined elsewhere in this report or come from the supporting operating systems.

The SRI Testbed uses the EUNICE operating system, which is a Berkeley UNIX[1] emulator for VAX computers using DEC's VMS operating system. EUNICE was developed at SRI to permit simultaneous access to UNIX and VMS software and system services, and to implement improvements to UNIX such as significantly faster image I/O. EUNICE is now a commercial product maintained by The Woolongong Group in Mountain View, California.

Some of the directory and file names were truncated for compatibility with an early EUNICE environment. (This is no longer necessary, although it may still be desirable for VMS compatibility.) The main program, subroutines, and help files are in directory /iu/tb/src/phoen; Major subdirectories are:

demo	- standard parameter sets;
display	- display routines;
do	- phase control scheduler;
flags	- flag parsing routines;
help	- help system text files;
include	- macro definition files;
k1	- command operators; ,
main	- PHOENIX main program;
misc	- I/O and misc. functions;
new	- new region maintenance;
queue	- queue maintenance routines;
v	- scheduling control functions.

To compile the PHOENIX program, just connect to this directory and type "make". You may type "make -n" to see what will happen if you do this. Additional options are documented in the header of the makefile.

Other major functions of the PHOENIX package have been moved to the /iu/tb/lib/visionlib histlib, intervlib, patchlib, and polygnlib directories because these subroutines may be of use to other programs. There is currently no documentation on these routines other than that in the source code headers.

Source code and help files for the CI driver are in /iu/tb/lib/cilib. For extensive documentation type "man ci" or run "vtroff -man /iu/tb/man/man3/ci.3c". The CI driver

[1]UNIX is a Trademark of Bell Laboratories.

uses command-line parsing routines in cilib/cmuarglib and in /iu/tb/lib/sublib/asklib; both of these may someday be replaced by the Testbed argument parsing routines in sublib/arglib.

Other utility routines contributed by CMU have been distributed to /iu/tb/lib/dsplib/gmrlib, /iu/tb/lib/imglib, and /iu/tb/lib/sublib, and are documented in /iu/tb/man/man3. Some of these have been modified or rewritten for the Testbed environment; the image access code, for instance, reads Testbed image headers as well as CMU image headers. Output map files are now created with Testbed headers.

One modification to PHOENIX was in the manual decision logic of the *fetch* phase, as controlled by the 'm' flag. The original code displayed the next region *after* the user had made a choice; the Testbed version now displays it before the choice is made.

SRI has also added a detailed display of the threshold selection heuristics during the *interval* phase. This is turned on by the 'H' flag, and takes effect if rundisplay is specified. Each feature histogram in turn is displayed in white. The thresholds before a heuristic takes effect are shown in blue; those remaining after the screening are shown in green. The user types a carriage return to proceed with the next heuristic. This integrates well with the retry facility for redoing the *histogram* or *interval* phases.

The original PHOENIX code assumed an upper-left origin for image and graphics display. CMU provided the conversion macros for changing image display to a lower-left origin as used on the Testbed. Since Testbed image format is also the inverse of the CMU format, the macros had the effect of displaying images right side up but in the lower-left corner of the screen.

Unfortunately the associated graphic displays did not use the coordinate conversion macros, and could not easily be made to do so. (Maintaining the original layout would require that all histograms and text be displayed upside down.) We have moved or interchanged some of the graphic components instead.

The original code limited rundisplay to images of 111 rows or fewer because of the multiquadrant display layout. With our altered layout, it was possible to extend this to 256 rows, although there is still a minor problem with text overwriting the images.

Modifications were needed in two of the threshold selection heuristics. The maxmin heuristic was rejecting nearly all thresholds if either of the outermost histogram bins held a large value; this was fixed by defining the outermost interval minima to be zero rather than the bin values. The absmin heuristic was rejecting all thresholds at nonzero bins if the global minimum was zero; this was fixed by clipping the global minimum to be at least one.

Several heuristic thresholds were permitted to take meaningless values (*e.g.*, **relarea** > 50 and **intsmax** = 1). These limits have been tightened. The **hsmooth** variable was originally limited to 20; we have extended it to 100. Several other arbitrary limits have been extended and default parameter values have been changed to the moderate values developed in Section 6. Some of the original and new defaults are:

splitmin:	1	-->	40
hsmooth:	1	-->	9
maxmin:	200	-->	160
absarea:	20	-->	10

relarea:	5	-->	2
height:	70	-->	20
absscore:	550	-->	700
relscore:	85	-->	80

Other minor changes included reducing debugging printout in clrscreen.c, adding printout of rejected feature set scores as well as accepted ones, changing the colors of some display elements, and fixing a bug in display of monochrome images. We have not yet removed a restriction against using red, green, or blue feature planes without using all three as segmenter inputs.

Several PHOENIX demonstrations have been set up in subdirectories of /iu/testbed/demo. The *chair* directory contains the original demonstration contributed by CMU: segmentation of an orange chair from a white background. The *show9* command shows the original red, green, and blue feature planes and the hue, saturation, intensity, y, i, and q feature planes computed with SRI's *convert* program. The *demo* command runs the interactive segmentation using only the red, green, and blue features. You may restore demo.ckp file to see the finished segmentation produced by shell file ckp.csh [using the original CMU parameter defaults].

The *portland* directory also has a *show9* command and a *demo* script that loads the final results for segmentation of the 512x512 *portland* image using strict and then moderate heuristics. You may run this script and then browse using the 'history', 'describe', 'display', and other informational commands. (Use the '*' and 'help *' commands to find out what is available.) You may also restore the mild.ckp file to see the effects of the mild (permissive) heuristics. This directory also has a *skydemo* script designed to show off PHOENIX as a skyline finder: just type 'Control-Z' or 'exit' to step to successive results.

The *demo* command in the *skyline* directory is very similar. It shows the results of segmenting a reduced *bishop* image using strict, then moderate, and finally mild heuristics.

Section 5

Program Documentation

This section constitutes a users' guide to the PHOENIX package as it is implemented on the SRI Image Understanding Testbed. As with any reference manual, it has occasionally been necessary to refer to terms before they are defined and discussed in detail. A preliminary scan through the section may be helpful on the initial reading. Additional information is available on-line, as described below.

5.1. Interactive Usage

The program requires one or more registered picture files as input. These typically represent red, green, and blue image planes, and perhaps intensity, hue, saturation, and other transformations as well. Texture planes may also be provided; they are not computed by PHOENIX. The program produces a region map, which is a picture file having 16-bit region numbers as the pixel values.

The set of input pictures is specified by a template and a *-f* flag followed by a set of feature keywords. For example:

```
phoenix /iu/tb/pic/chair/4.img -f red green blue ...
```

specifies that the files 4red.img, 4green.img, and 4blue.img in the directory /iu/tb/pic/chair are to be used as the input pictures.

Once started, the user typically sets some flag values to control the scheduling process and display options, then issues the *segment* command. This begins segmentation of the image, which will continue to the halting point specified by the A, B, and C flags (see below). The user may also interrupt processing with the 'Control-C' key, and may then examine or alter the current status.

Segmentation is normally done by depth level, with all regions at one depth segmented before any of their subregions are processed. The segmentation of a single region at a single depth constitutes a *pass*, and consists of a region-dependent sequence of various *phases*.

5.1.1. Invocation Options

The following options may be specified on the initial command line. All other commands must be typed in interactively or piped in using a batch script. (See Section 5.2.)

-e Echo commands as they are read from a file. If this is not specified, initialization commands will execute invisibly. Interactive script commands invoked with '<' are not affected by this flag.

-f feature ...
Feature plane specifications as illustrated above. See [Clark81] for a full description of the picture naming system.

-i file
-I file
Read initialization commands from *file* before accepting commands from the terminal. Only one such file may be specified. The *-I* form exits without accepting terminal input.

-o file
-O file
This mandatory parameter specifies the output map file. The *-o* form will create a new file; if it already exists, PHOENIX will ask whether you want to overwrite it. The *-O* form will open an existing map file.

-r region#
-R region#
These two parameters are used with existing (*-O*) map files to specify the current (*-r*) region for further segmentation and the highest (*-R*) region number to be updated.

-s Execute a single *segment* command and then exit. This is usually combined with initialization commands (see *-i*) to set 'flags = ABC' for continuous segmentation and perhaps 'flags = q' to squelch tty output.

5.1.2. Interactive Commands

PHOENIX is implemented using the CI command interpreter. Type '?' for a list of commands that are available from the CI driver itself. Most of these have to do with interactive help facilities.

The following PHOENIX commands can be entered from the keyboard at any stopping point during the session. Arguments that are not specified as part of the command will be requested.

abort
Terminate segmentation of the current region. The region is put back at the head of the segmentation queue.

checkpoint datafile mapfile
Save the current state of the segmentation. Global variables are stored in *datafile* and a copy of the region map is written to *mapfile*. Histograms, interval sets, and other temporary data structures are not saved.

If you specify a nonexistent directory, you are asked whether it should be created. If either of the specified files already exists, you are asked whether you want to overwrite the file or specify a new one. Simply typing a carriage return will abort the command.

The datafile contains a reference to the mapfile name, so you may not use operating system commands to rename this file. (You may move both files to a new directory as long as the same name is used.) The best way to rename checkpoint files is to restore them and then write them out again under new names.

clear {s|t}
Remove all regions from a specified queue. You are asked whether you really want to do this.

describe [type] [identifier]
Describe a data structure. You must specify the type of object and which one you want. Currently you can ask about regions, histograms, and interval sets. Regions are identified by number; histograms and interval sets by feature. You may specify 'all' features if you wish.

display [type] [identifier]
Display an object or data structure. You may display the image, particular regions, or the current segmentation map overlay. You may also display the current region histograms or interval sets during phases when they exist. All of these displays temporarily erase any rundisplay output.

dqueue {s|t} howmany
Remove *howmany* regions from the head of a queue.

exit Terminate the PHOENIX session. The region map is properly closed before exiting.

history [region#]
Print the history of a region. Describes the region's ancestors, beginning with the earliest.

list {s|t} [howmany] [nth]
List the elements of a queue. Lists *howmany* elements of the specified queue, starting with the nth element from the head of the queue if nth is positive, from the tail if negative. In either case the elements are listed starting with the one nearest the head of the queue.

prune [region#]
Prune a portion of the segmentation tree. Moves the specified region back to the head of the segmentation queue, deleting all of its descendents from the region tree and from the output region map. You are asked whether you really want to do this.

queue {s|t} region# [region# ...]
Add regions to a queue. Adds the indicated regions, one at a time, to the head of the specified queue. They will appear on the queue in reverse order, i.e., the last one listed will be at the head of the queue. No check is made for duplicate regions or for segmented regions being added to the segmentation queue.

If you accidentally invoke this command (*e.g.*, while trying to *quit*), just specify 0 for the region number.

release
Relinquish the display.

restore datafile
Restore a checkpoint file (which must correspond to the current image, but may have different feature planes). The state existing when the checkpoint was taken is restored, except that actions following the last *collect* phase will be forgotten and the display is not restored. You cannot restore a checkpoint if you are running PHOENIX in a different directory from when you created the files.

retry phase
Re-execute a previous segmentation phase. This is only valid when in the middle of segmenting a region. Data structures created since the previous start of the indicated phase are deleted. Table 4.1 shows the transitions that are permitted; current states appear on the left and desired states along the top.

segment
Run the next segmentation phase. The next scheduled phase of segmentation will be executed.

transfer {s|t} howmany
Transfer regions from one queue to the other. Moves *howmany* elements from the head of the indicated queue to the head of the other queue, one at a time. Hence the elements will end up in reversed order. No check is made for duplicate regions.

Table 4.1. Legal Retry Transitions

	ftch	hist	intv	good	next	thrs	ptch	eval	slct	clct
fetch	-	-	-	-	-	-	-	-	-	-
histogram	-	Y	-	-	-	-	-	-	-	-
interval	-	Y	Y	-	-	-	-	-	-	-
goodfeatures	-	Y	Y	Y	-	-	-	-	-	-
nextfeature	-	Y	Y	Y	-	-	-	-	-	-
threshold	-	Y	Y	Y	-	Y	-	-	-	-
patch	-	Y	Y	Y	-	Y	Y	-	-	-
evaluate	-	Y	Y	Y	-	Y	Y	Y	-	-
selection	-	Y	Y	Y	-	-	-	-	Y	-
collect	-	-	-	-	-	-	-	-	-	-

5.1.3. Execution Phases

Interaction with PHOENIX is normally through the control of the *execution phases*. These are "packets" of executable code that together constitute the *segment* command. The normal sequence of segmentation phases is illustrated in Figure 5.1. The user may interrupt the segmentation at any time, either at scheduled interrupts (see flags A, B, and C) or by using the 'Control-C' or 'delete' keys. Execution resumes when another *segment* command is issued.

The phases, in order of normal execution, are listed below. (The descriptions assume that rundisplay has been set to *yes*; otherwise displays must be requested using the *display* command.) To begin or continue this phase sequence, issue the **segment** command. The phases that are then run depend on the control flags which have been set.

fetch
: The next region is fetched from the segmentation queue. (Initially the entire image is one region.) If the 'd' flag is set, a description of the region is printed. The region is expanded by pixel replication and is displayed above the original image (where the region center is marked by the cursor). If the region has an area less than splitmin pixels, or if the 'm' flag is set and the user declines the region, it is declared terminal and a *collect* phase is scheduled. Otherwise the region is passed to the *histogram* phase. You will not be allowed to resegment a region that has already been segmented.

histogram
: A region histogram for each color or feature is computed. Each histogram is smoothed using an unweighted moving average if the hsmooth variable is set greater than 1.

interval
: Each feature histogram is broken into intervals (as described below) and is displayed with the thresholds marked in red. An interval-set quality measure is computed for each feature and boxes are drawn around histograms with acceptable scores. If none was acceptable, the region is declared terminal and a *collect* phase is scheduled; otherwise (if the 'd' flag is set) the interval-set score for each feature is printed.

goodfeatures
: This initializes the spatial evaluation loop [phases *nextfeature* to *evaluate*; see Figure 5.1]. If there are no candidate features, the region is declared terminal and a *collect* phase is scheduled.

nextfeature
: The next good feature is chosen. If all have been evaluated, a *selection* phase is scheduled.

threshold
: The region is thresholded (or level-sliced) using the chosen interval set, and is displayed with a different intensity for each interval. Corresponding intensities are indicated on the feature histogram. The connected components are outlined on the expanded original and

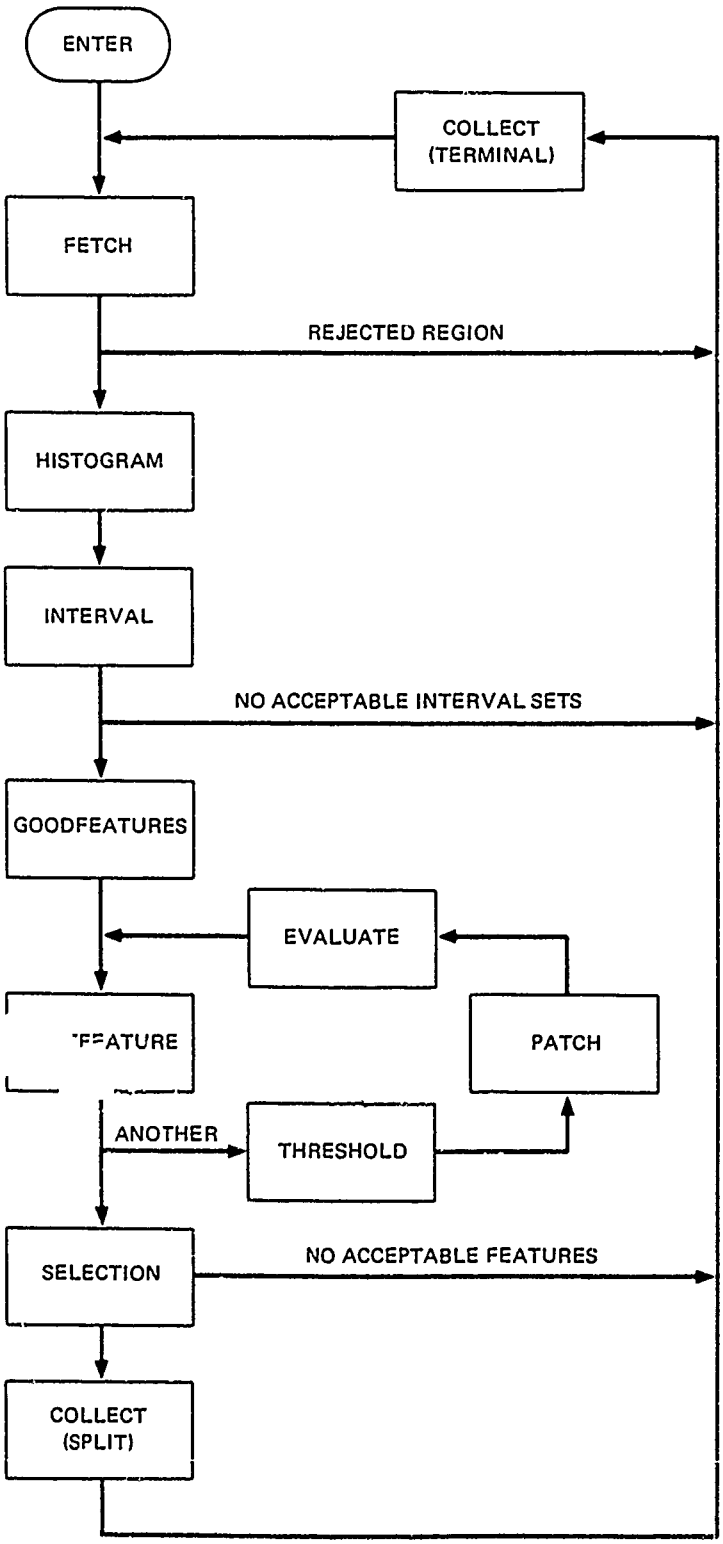

FIGURE 5.1 SEGMENTATION PHASE SEQUENCE

thresholded region images. (This outlining is separate from the boundary extraction done in the following *patch* and *collect* phases.)

patch
: Connected components for each intensity interval are extracted. These components are stored as *patches*, which are run-coded representations together with a few shape descriptors (linear dimensions, area, centroid, number of holes, *etc.*). Patches in the foreground (selected intensity) are 4-connected, while the corresponding background is considered 8-connected.

evaluate
: Patches are classified as either valid regions or noise regions. At present this is determined by comparing the patch area to the **noise** threshold, without regard to patch shape. Each noise region is marked with a dot in both the expanded original image and the thresholded image. The feature is then evaluated by computing the percentage of noise area over the whole image. A *nextfeature* phase is always scheduled to follow this one.

selection
: When all features have been evaluated, the one with the least noise area is selected for segmenting the region. A feature is disqualified if the noise area exceeds the **retain** threshold, or if any one of its intervals failed to produce a valid patch. If no suitable feature is found, the original region is declared terminal; in either case a *collect* phase is scheduled next.

collect
: If the original region has been declared terminal, it is moved to the head of the terminal queue. Otherwise the valid patches (merged with their contained noise patches) are converted to regions. This involves computing the polygon boundaries of the new regions, updating the history list, adding the regions to the segmentation queue, inserting them in the stored region map, and drawing them on the original image display. (These outlines accumulate so that the overlay on the original image always represents the current state of the segmentation. If the user edits the segmentation history or asks for other displays, the outlines may not correspond to the full segmentation.)

This order of execution may be altered in several ways. If an error occurs, *e.g.*, a memory allocation failure, the same phase will be rescheduled as the next phase. The user may also interrupt processing and attempt to schedule a previous phase with the retry command. The system permits some retries and forbids others, depending on the last completed phase and the next scheduled phase. It will object if either a *fetch* phase or the phase you specify is already scheduled next, or if you try to jump forward in the phase sequence. It will also object if you try to jump into the middle of a loop, in particular, you may not retry a *nextfeature* phase.

The segmentation stops when there are no more regions on the segmentation queue. The user may then (or at any time before) ask for various displays and information, edit the segmentation, or save the current region map and region description file. A saved state may be reloaded later and processing may continue.

5.1.4. Status Variables

PHOENIX maintains a set of read-only variables or status query commands. To query the value just type the name of the variable. Although the values may not be set directly, some of them may be changed by other PHOENIX commands.

features
: Features currently being used in segmentation. This is just the list of names following the *-f* flag on the initial command line.

images
: Picture files being used in the segmentation.

lastphase
: Last segmentation phase completed.

nextphase
: Next segmentation phase to be run.

phases
: A list of all the segmentation phases. The last and next phases are designated.

regions
: The number and range of existing regions.

time
: Real time and CPU time spent in each phase and in the entire PHOENIX run. (Real time for a restored segmentation is not meaningful.)

See also the execution flags and control variables documented below.

5.1.5. Execution Flags

Flags (on/off variables) may be used to control execution of the entire program or of any phase. Local phase flags take precedence over global flag settings.

To find out what flags are set, type *flags*. You may turn off all flags by typing *flags = -**. To selectively turn flags on and off, use a command like *flags = -AB+g*, where the plus sign may be omitted if there is no preceding minus sign. The following flags are available:

A (default)
: Begin the next non-*fetch* phase without interrupt. This permits the current *segmentation pass* on the current region to run to completion.

B
: Permit same-depth *fetch* phases without interrupt. Segmentation of the current regions will run to completion, but their children will not be segmented until the next *segment* command is given. Flag B is only meaningful if flag A is set.

C Permit *fetch* phases that initiate new levels of segmentation. Segmentation of the entire image will run to completion. This is only meaningful if flags A and B are set.

D Enable debug printout. (See also flag G.) This option turns on printout of storage management messages.

G Print display subroutine entry, exit, and debugging messages.

H If rundisplay is turned on, step through a detailed display of threshold selection heuristics during the *interval* phase. [This is an SRI addition.]

P Pause rather than stop on interrupts caused by having flags A, B, or C turned off. (To continue after a pause, type a carriage return. To continue after a stop, type *seg*[*ment*].)

d Describe fetched regions and feature quality statistics.

g Order regions on the queues globally by area. This overrides the 'o' flag.

m Request a manual decision on whether to further segment a fetched region. If rundisplay is active, the region will be displayed. The 'd' flag should usually be set so that there is a further basis for the choice.

o (default)
Order regions by area within each depth. If neither this nor the 'g' flag are set, regions are simply added to the tail of the segmentation queue as they are generated.

q Execute quietly, without normal tty output. This does not affect output due to the 'G' or 'd' flags, nor echoing of prompts and commands.

v (default)
Autoverbose mode. Run in verbose (as opposed to quiet) mode for regions with area greater than autoarea. This has precedence over the 'q' flag, but only takes effect locally during a *collect* phase and then permanently during a *fetch* phase.

To set a local phase flag, use a command of the form *during <phase> = -*+AB*. This string will be used to modify the flags variable during the specified phase.

To examine the current modifier value, type *during <phase>*. There is currently no command to display all of the local flag settings at once.

Resetting (disabling) a "during <phase>" option is a little difficult. There is no command to reset all of the phase modifiers at once. To reset them individually you should specify "during <phase> = +".

5.1.6. Control Variables

The user displays and decision logic used by PHOENIX may be fine tuned by setting various option variables and thresholds. To turn on the *rundisplay* option, for instance, type *rundisplay = yes*. To ask for the current value, just type *rundisplay*. Abbreviations are accepted.

The following affect the *fetch* phase or the PHOENIX session as a whole. Default values are listed in parentheses.

autoarea (0)
: Maximum region area for the autoverbose option (flag v) to select quite mode.

depth (infinite)
: Maximum depth of the segmentation tree. Regions at this depth will not be split further. (This test is currently made at the end of the *collect* phase.)

rundisplay (no)
: Use a real-time multiquadrant presentation of processing results. This cannot be used for images larger than 128x128.

  The original image is displayed in the lower-left quadrant with all region boundaries overlayed in red and the cursor centered in the current region. A window containing the current region is expanded by pixel replication and displayed in the upper-left quadrant. Histograms and interval sets are displayed along the right side of the screen. During spatial analysis, the lower-right quadrant contains the selected histogram and the upper-right quadrant displays the thresholded region window. Patches are outlined in green in both of the expanded windows, and noise regions are marked by blue dots.

splitmin (40)
: Minimum area for a region to be automatically considered for splitting. This is an absolute area, not a percentage of the image area.

A fetched region is first histogrammed, and each feature histogram is smoothed. This is controlled by

hsmooth (9)
: Histogram smoothing window. Smoothing is done with an unweighted moving average; the outermost bin values are assumed replicated beyond the ends of the histogram.

The heart of the PHOENIX system is the *interval* phase, since histogram segmentation is the major step in color image segmentation. The variables that control this process, along with their default values, are listed below in the order of their application.

maxmin (160)
Lowest acceptable peak-to-valley-height ratio expressed as a percentage.

absarea (10)
Minimum area for an interval to be retained.

relarea (2)
Minimum acceptable percentage of total histogram area.

height (20)
Minimum peak height as a percentage of the second-highest peak. This test is skipped if there are only two intervals.

absmin (10)
Maximum retained valley height as a multiple of the lowest (or "absolute minimum") valley in the histogram. Intervals separated by higher valleys will be merged. This test is skipped if there are only two intervals.

intsmax (2)
Maximum number of intervals in each final interval set. The intervals will be reduced to this number by merging (*i.e.*, eliminating histogram cutpoints), starting with the highest valley.

Each interval set containing more than one interval is then assigned a score:

$$1000\ \frac{\textit{peak height} - \textit{higher shoulder}}{\textit{peak height}}$$

Interval sets with low scores are not considered for spatial analysis. Thresholds used in the spatial evaluation, or *goodfeatures* to *evaluate* phases, are:

absscore (700)
Minimum acceptable interval set score. Less promising interval sets will not be selected for spatial evaluation.

relscore (80)
Minimum acceptable percentage of the highest set score. Features with lower interval set scores will not be considered.

isetsmax (3)
Maximum number of interval sets (features) that will be evaluated.

noise (10)
Minimum area of noise regions. Only regions larger than this are retained.

The following affect the *selection* and *collect* phases:

retain (20)
Maximum acceptable noise area as a percentage of a region's total area. Regions with more noise content will not be retained.

tolerance (0.1)
Tolerance for polygon fitting. This affects only the output description and has nothing to do with the region-splitting algorithm.

5.2. Batch Execution

The PHOENIX program offers two methods of invoking prestored commands. The first is the invocation of CI command files, either interactively or with the *-i* command-line flag. For example, you might give the command

```
> <chair.cmd
```

where the file chair.cmd contains the commands

```
flags = -BC+APdov
rundisplay = yes
```

In this case the PHOENIX program will set the flags for a moderately interactive session with the special *rundisplay* turned on.

The second method is to drive the entire PHOENIX session from an operating system script. A UNIX C-shell script might look like:

```
# PHOENIX segmentation system.
# Supply the image name as an argument.

rm -f $1.map
phoenix /iu/tb/pic/$1/.img -f red green blue
    -o $1.map -i $1.cmd <<!
  flags = ABCPdov
  depth = 4
  rundisplay = no
  segment
!
echo "Finished."
```

This script is designed to run without user interaction or visible displays. It does print some information during processing, but does not wait for you to look at it. (You can temporarily halt the processing if your terminal accepts *hold* or *^S^Q* handshaking commands.)

To save the typed terminal output you should pipe the standard output to a file. The UNIX method for doing this is to add *>session.log* to the *phoenix* command within the script or to the UNIX command line that invokes the script. You may also use the UNIX *script* or *tee* commands to route the typed output to a file and to your terminal.

The actual submission of this shell script is described in the UNIX Programmer's Manual. You should run it in foreground mode if you want to interact with the program. If you run it in background mode, be sure to pipe the output to a log file so that it won't appear on your terminal. You can monitor the log file during execution

(using the *cat* or *trail -f* commands) to make sure everything is running smoothly, although the log file will typically run somewhat behind the actual program execution. You can also halt the process or reconnect it to your terminal if you wish.

Section 6

Evaluation

This section documents the performance of the PHOENIX program in test runs on a variety of imagery. Rules are given for setting various scene-dependent parameters, and performance characteristics are evaluated. The section ends with an application of PHOENIX to the problem of skyline delineation.

6.1. Parameter Settings

PHOENIX is a moderately complex system with numerous execution options and 14 user-settable variables that control the segmentation process itself. We will describe the effects of each option alone and in combination with others.

In addition, we will describe the threshold variable settings for "mild", "moderate", and "strict" screening of potential feature thresholds. These correspond to permissive, moderate, and cautious segmentations. These three categories reduce the 14 variables to a manageable single parameter.

Also listed are the minimum and maximum legal values for the SRI version of PHOENIX; the "disabled" value turns off a heuristic completely, and a "drastic" value makes it so strict that very few histogram cutpoints will get through.

We recommend that PHOENIX command files be used as a mechanism for quickly loading sets of commands. We have used files named *strict.cmd*, *moderate.cmd*, and *mild.cmd* in directory /iu/tb/src/phoenix to store the corresponding 14 threshold settings (with the exception that **intsmax** is always set to 2). Files named *run.cmd*, *tst.cmd*, and *display.cmd* store commonly used flag settings and control variables. Each user should develop such command files for the tasks he commonly performs. (PHOENIX should also permit a directory search path to be specified so that standard files could be used or selectively overridden. The underlying CI driver supports this.)

Flags

The flag mechanism controls the amount of interaction between the user and the system. Some flags tell the scheduler whether to proceed autonomously or to stop and ask for commands; others control verbose printout and debugging messages. Several sets of flags (*e.g.*, 'ABC' or 'go') might be better represented by single variables than by interacting flags, but the flag mechanism is useful for allowing "during <phase>" control.

These control options are reasonably straightforward; which flags you should set depends upon what you want to do. They do not affect the segmentation algorithm, so there is no danger of setting them "incorrectly."

Evaluation

The 'm,' 'g,' and 'o' flags come closest to affecting the segmentation process. The 'm' flag allows you to override the *splitmin* heuristic and decide manually whether each region should be split further. This is a valuable option, although a time-consuming one. There is also a need for a more general facility that would accept arbitrary selection criteria for transferring regions from one queue to another. (Although specific tests can be added to the current C-based driver, it would be much easier to implement such screening in a LISP-based driver language.)

The 'g' and 'o' flags control the order in which newly-created regions are added to the segmentation queue. If flag g is set, the regions are ordered globally by size. If 'o' is set (and 'g' is not) the regions are ordered by size within each segmentation depth. If neither is set, new regions are simply added sequentially to the tail of the queue. (There is no provision for resorting the queue when you switch from one to another of these options. Either this should be implemented or the queues should be unordered with selection done during the *fetch* phase.)

Global ordering by size is useful for interactive sessions. The segmenter begins with the largest region and keeps whittling off small subregions until the large region is homogeneous. Then it picks the largest subregion and does the same. With this method there is enough continuity so that you can keep track of what is happening. It would also be good for cueing applications where it is important to find small "blips" quickly.

Depth ordering by size is more useful for automatic segmentation. It has the property that an interrupted session provides a good partial segmentation into regions of similar prominence. If run to completion, the segmentation is identical to that produced with global ordering.

Global ordering by size is equivalent to a depth-first search through the segmentation tree, whereas the other two options (depth ordering by size and sequential ordering) are breadth-first searches. There is a need for more flexible best-first ordering, where the sorting criterion could be based on region shape, color, position, or other properties.

A final note: we suggest that the command "flags = A" should reset all flags other than A, instead of adding A to the current flag list. The "flags = -*+A" syntax could then be used to reset all the "during <phase>" flags as well as the global flags.

Rundisplay (no)

Rundisplay can take only two values: 'yes' or 'no.' It controls the special interactive display that is useful for exploring the system and for debugging. It is so useful, in fact, that any production version of the system should be extended to include some type of rundisplay even for images larger than 256x256.

Rundisplay allows the logic flow to be followed step by step. This has been extended by SRI (*via* the 'H' flag) to include the action of each heuristic cutpoint screening. It could be extended even further to include separate display of heuristics that are currently combined, such as the **absarea** and **relarea** or **abscore** and **relscore** procedures. On the whole, though, the current facility is excellent.

Autoarea (0)

Autoarea controls the size of region for which verbose printout is used if the 'v' flag is set. Normally this variable will be left at its default value of zero. This has no effect on the segmentation algorithm.

Depth (infinite)

```
Disabled:       INF   [Also disabled by flag `g'.]
Mild:            20
Moderate:        10
Strict:           4
Drastic:          1
```

Depth is active when depth ordering of the segmentation queue is used. It prevents segmentations of regions lower than the specified depth in the segmentation tree. (Larger numbers refer to lower depths.) The depth limit can be used to restrict processing time, although this could be better achieved with the **splitmin** threshold or with an actual threshold on time spent.

It is difficult to see how this parameter can be used effectively. Recursive segmentation depth is not a property of a region, but of the region and its context. A strict depth limit will cause differing segmentations of a region when differing orders of features are used to extract it from its background. We therefore recommend that this variable always be disabled or left at the mild setting.

Splitmin (40)

```
Disabled:         1
Mild:            20
Moderate:        40
Strict:         200
Drastic:        INF
```

Splitmin is the only control, other than using **depth** or direct manipulation of the segmentation queue, for which fetched regions are to be segmented further. Any region smaller than **splitmin** is declared terminal and is moved to the terminal, or 't,' queue. (This is useful for examining all regions. Just set **splitmin** to a very large number, turn on rundisplay, optionally set the 'd' flag, and begin segmentation. Each region will be displayed and described before it is rejected.)

The heuristic thresholds given above seem reasonable, but **splitmin** should really be determined by the size of target or object facets being sought. It should be at least twice the **absarea** and **noise** thresholds.

A second heuristic might be used to limit regions to a specified fraction of the image area, thus permitting consistent segmentation across different imaging resolutions. In fact, a more general screening facility could be implemented (particularly in a LISP-based driver) for selecting regions by shape, color, position, orientation, or other characteristic.

Hsmooth (9)

Disabled:	1
Mild:	5
Moderate:	9
Strict:	25
Drastic:	100

Hsmooth is the width of the averaging window used to smooth each feature histogram. (Any spatial smoothing of the feature planes themselves is outside the province of PHOENIX. Such smoothing combats the breakup of textured regions.) Histogram smoothing eliminates many false cutpoints that are due to texture, digitization effects, or color transformations. It also improves the reliability of several other heuristics, as described below. The amount of smoothing required is often quite large because PHOENIX has difficulty distinguishing even small notches from broad valleys between peaks.

Histogram smoothing is done with an unweighted moving average computed by replicating the outermost bin values to plus and minus infinity. This is simple to implement, but may introduce artificial peaks when used on small regions with scattered histogram values. A center-weighted moving average would have better filter characteristics.

This smoothing turns out to be very important – and different values are required at different times. Strict smoothing can be used on peaks that are well separated. This simplifies the task of later heuristics, although cutpoint placement is not critical in such cases. Strict smoothing would be useful for properly splitting peaks that overlap slightly, but would cause the **maxmin** heuristics to discard the cutpoint altogether; moderate or even mild smoothing must be substituted. Mild smoothing is also required for finding small regions within large ones, but is insufficient for segmenting the noisy histogram of a small region.

The problem is that PHOENIX does not use explicit models of histogram peaks. It considers only very simple statistics of histogram intervals, such as apex and shoulder heights. It has no notion of valley width: all heuristics treat a single-bin notch as being identical to a very wide valley. Histogram smoothing is the only mechanism in PHOENIX for making such a distinction, and it is insufficient for the task.

Although modeling of histogram peaks and valleys is the best solution, some improvement could still be made in the histogram smoothing mechanism. A smoothed histogram should augment the original, not replace it. Each heuristic should be able to apply the smoothing that it requires. Then mild smoothing could be used for selection of the initial cutpoints and strict smoothing could be used for positioning of a final cutpoint.

Maxmin (160)

Disabled:	100
Mild:	130
Moderate:	160
Strict:	300
Drastic:	10000

Maxmin is the minimum acceptable ratio of apex height to higher shoulder. Any

interval failing this test is merged with the neighbor on the side of the higher shoulder. The test is then repeated on the combined interval. The overall effect on a set of cutpoints is to eliminate those that are on the sides or tops of major peaks.

The original version of PHOENIX had difficulty if an apex abutted either end of the histogram. The outer shoulder height was taken to be the apex height, and the interval would fail the **maxmin** test. Further, the merged interval would inherit this shoulder height and would also fail the test. This process continued until all intervals had been rejected. We have fixed this in the SRI version by assigning an outer shoulder height of zero to the outermost intervals; this represents the bin height at plus or minus infinity.

Maxmin is a powerful heuristic. With strict smoothing and all other heuristics disabled, **maxmin** alone is able to produce reasonable segmentations. It is even more powerful when combined with the area heuristics. With mild or moderate smoothing, **maxmin** passes clusters of cutpoints in the noise regions between major peaks. This is fine if the clusters can be thinned by the **absarea** and **relarea** heuristics, but a poor selection may be made if they are left for the **intsmax** heuristic.

The problem here is that PHOENIX has no "quality" score for histogram valleys. It assumes that cutpoint bin height is an adequate measure, whereas width and depth relative to the neighboring peaks are also important. PHOENIX can only incorporate such knowledge by smoothing the histogram, and the amount of smoothing required depends on how separated the peaks are.

Absarea (10)

```
Disabled:        1
Mild:            5
Moderate:       10
Strict:        100
Drastic:       INF
```

Absarea is the minimum histogram area that a usable interval may contain. It should usually be set to the same value as the **noise** threshold. (Perhaps the two thresholds should be combined.)

This threshold is tied to the pixel resolution, and so will cause differing effects in images of differing resolution. The value really depends on the size of objects you are trying to find, and on the number of pieces that such an object might be broken into by texture characteristics.

Relarea (2)

```
Disabled:        0
Mild:            1
Moderate:        2
Strict:         10
Drastic:        50    [30 is very strict.]
```

Relarea is the minimum percentage of the histogram area that a usable interval may contain. This intended to eliminate noise peaks (PHOENIX has no explicit model of histogram noise statistics) and to conserve processing time by skipping doubtful intervals.

Evaluation

This is a questionable heuristic since the effect on a particular interval depends on the total area in peaks that may be quite distant. Small peaks are best skipped if larger ones are available in any feature, but there are times when segmentation must be done on the small peaks or not at all. If CPU time is not a problem, it is best to pass these small intervals on to other heuristics and to spatial analysis.

Absarea and **relarea** should both be reduced slightly to allow for PHOENIX's tendency to clip the tails of major peaks. (This is due to the lack of a statistical or semantic model for histogram peaks. A notch in the tail of a major peak is treated the same as a wide valley, and the area heuristics often merge the clipped tail to the wrong side.) Small thresholds for the area heuristics allow multiple cutpoints to survive for screening by the later heuristics.

Height (20)

Disabled:	**0**
Mild:	**10**
Moderate:	**20**
Strict:	**50**
Drastic:	**100**

Height is the minimum acceptable apex height as a percentage of the second highest apex. (The test is skipped if there are fewer than three intervals.) Cutpoints between the highest histogram peaks are favored over those isolating low or noise peaks.

This is a questionable heuristic, for much the same reasons as **relarea**. It is difficult to choose a reasonable value because peak height is much less important than the separation between peaks. Further, the effect on a particular interval can depend upon distant peaks in the same histogram.

The effect can seem mysterious when the second-highest apex is not readily apparent. With mild smoothing the second-highest apex is often part of the main histogram peak separated by a small notch. The **height** heuristic then tends to eliminate all cutpoints that are not similar notches high on major peaks. A strict **maxmin** threshold can combat this by pushing secondary apexes down the side of the main peak. Strict smoothing can also be used to eliminate the notches, although the amount of smoothing needed varies with the histogram characteristics. The simplest solution is to simply disable this heuristic or use a very low threshold.

Absmin (10)

Disabled:	**1000**
Mild:	**30**
Moderate:	**10**
Strict:	**2**
Drastic:	**1**

Absmin screens cutpoints rather than interval statistics. It is the lowest acceptable multiple of the minimum cutpoint bin height. (The test is skipped if there are less than three intervals.) An interval is rejected if either shoulder is not at least **absmin** times the height of the lowest cutpoint bin in the histogram. Unfortunately the name of the heuristic does not make this clear.

The use of a multiplication factor (or ratio) as a threshold entails some difficulties. Unless strict smoothing is used, the global minimum is often zero. All cutpoints with nonzero bin heights are then rejected, and frequently only the global minimum itself will survive. There was no setting of **absmin** that would disable this behavior. We have therefore modified the ratio test in the SRI version so that the denominator is always at least one.

The heuristic is still unstable near zero, but is tolerable and perhaps even useful for large regions. A mild or moderate threshold tends to pass clusters of cuts in the valleys unless they have been thinned by the preceding area heuristics. A strict threshold performs surprisingly well all by itself; with strict smoothing there will be only one cutpoint in a valley, and with mild smoothing there is often a noise notch deep enough to eliminate the other cutpoints.

For small regions (under 100 pixels) this heuristic is useless. Cutpoints for these histograms are nearly always at zero height, so this heuristic cannot choose between them.

Intsmax (2)

Disabled:	**100**
Mild:	**6**
Moderate:	**3**
Strict:	**2**
Drastic:	**2**

Intsmax is the maximum number of intervals permitted in the final interval set for a feature. If more intervals reach this point, the one with the highest maxmin ratio (apex to higher shoulder) is merged with the neighbor on the higher-shoulder side. This process continues until the desired number is reached.

The effect depends on the number and nature of cutpoints passed by the previous heuristics. It tends to favor cutpoints in valleys because small amounts of noise produce high maxmin ratios. This behavior is reasonable, although for mild smoothing it favors noise notches over the centers of broad valleys. (Actually PHOENIX has no notion of the center of a valley. If given a flat valley, it will put the cutpoint on the leftmost bin. Only noise notches or high smoothing will pull the cutpoint to the center.) **Intsmax** may also pass a cluster of cutpoints in one valley in preference to cutpoints scattered through many valleys. The area heuristics may be used to combat this.

Multiple cutpoints can be investigated either in one segmentation phase of many intervals or in many phases of two intervals each. The former saves considerable computation, but gives poor results for reasons described in the **noise** section. It is best to set **intsmax** to two unless there is need to conserve computational resources.

Absscore (700)

Disabled:	0
Mild:	600
Moderate:	700
Strict:	930
Drastic:	1000

Absscore is the lowest interval set score that will be passed to the *threshold* phase. The score is currently just the maximum over the interval set of all the apex minus higher shoulder to higher shoulder ratios, which is equivalent to the maxmin ratio.

This heuristic partially duplicates the screening performed by the **maxmin** threshold, and should be coordinated with that value. The conversion formulas for the component ratios are

$$interval\ score = 1000 - \frac{100000}{maxmin}$$

$$maxmin = \frac{100000}{1000 - interval\ score}$$

It would be simpler if the maxmin ratio were used throughout.

Unfortunately this simple score is poorly suited to choosing a good interval set: one that will generate a segmentation with very few noise regions. Noise regions are a symptom of the worst threshold for an interval set, whereas this formula uses the best threshold. The minimum over the interval set would thus be more appropriate, although an area-weighted average might be better.

An even better score would consider peak and/or valley shapes. The current score is a very weak model, as can be seen for the case of an interval that contain several small histogram peaks: the ratioed apex and shoulder heights may belong to different peaks. The current score is also useless on small regions (*e.g.*, 100 pixels) since the cutpoints usually have zero height and every interval set has a perfect score of 1000.

Relscore (80)

Disabled:	0	
Mild:	65	
Moderate:	80	
Strict:	95	
Drastic:	100	[Single best score is verified.]

Relscore is the least percentage of the highest interval set score that will be passed to the *threshold* phase. This is intended to eliminate poor features when better ones are available, but is less effective for this than the **isetsmax** heuristic.

The difficulty arises because a very small peak separated by zeros will have a perfect score of 1000. (This becomes more likely with small regions or mild heuristics.) Other features will then be rejected if not within **relscore** of 1000, so that **relscore** is acting much like **absscore.** If the small peak is finally rejected by spatial analysis, the region is declared terminal and the other features are never tried. To prevent this, either use strict area heuristics or a very mild **relscore** of approximately one-tenth **absscore.**

Isetsmax (3)

```
Disabled:       100
Mild:             5
Moderate:         3
Strict:           2
Drastic:          1    [Single best score is verified.]
```

Isetsmax is the maximum number of interval sets (features) to be passed to the *threshold* phase. If more than this have survived screening, the **isetsmax** with the highest scores will be chosen. Rejected features will get a second chance in later PHOENIX passes only if one of the chosen features succeeds in segmenting the region.

Noise (10)

```
Disabled:         0    [Always segment on first feature.]
Mild:             5
Moderate:        10
Strict:          50
Drastic:      10000
```

Noise is the size of the largest area that is to be considered noise. This heuristic is applied after thresholding and connected-component extraction. Patches larger than **noise** pixels will be retained; others will be merged with surrounding regions. Note the similarity of this behavior with that of rejecting a cutpoint with the **absarea** threshold.

This is a very difficult threshold to set because the size of noise regions is dependent on the task, the object, and the image resolution. It might be worthwhile to add a relative noise heuristic that would judge the patch area in relation to the original region area. This capability is partially available through the **relarea** heuristic.

Even better would be a noise score or set of region rejection heuristics that would consider boundary shape, contrast with surrounding regions, local noise statistics, and task-dependent semantic information.

Retain (20)

```
Disabled:       100
Mild:            40
Moderate:        20
Strict:           4
Drastic:          0
```

Retain is the maximum percentage of the original region area that may consist of merged noise regions. If the total noise area exceeds this, the feature will be rejected. It will also be rejected if any interval produces only noise patches, regardless of the noise percentage. After all interval sets have been tested, the one with the least noise area is selected for final conversion of patches to new regions. If two regions are tied, the first is chosen arbitrarily. (This is the only place where the input order of the features makes a difference.)

This heuristic is not intelligent enough for the burden placed upon it. It should be favoring large, compact areas (or other target shapes) as well as noiseless ones. At

present it is quite happy with a trivial segmentation of a tiny region vs. all the rest. This is fine for cueing applications, but poor for general use.

Use of multiple cutpoints (*i.e.*, **maxints** greater than 2) introduces additional noise and increases the likelihood that some interval will fail to produce a good patch. PHOENIX is unable to recover from this by deleting one cutpoint at a time or by retaining the good patches and discarding the rest.

One solution is to add relative noise heuristics as well as this absolute one. Noise could be expressed as a percentage of patch area: any patch containing too much noise would be rejected. It could also be expressed as a percentage of interval area. Either of these would integrate well with retention of all patches or intervals that are useful, without regard to the success of the feature as a whole.

Even better would be a set of region-acceptance heuristics that would consider boundary shape, contrast with surrounding regions, local noise statistics, and task-dependent semantic information. Such heuristics would be easiest to implement in a LISP-based driver.

6.2. Performance Statistics

To further evaluate PHOENIX, it is necessary to choose a task domain. We have selected skyline delineation. This is the problem of determining the skyline in an image that includes both ground and sky.

It should be noted that this problem is not always well defined. Images of cloud-shrouded mountain peaks or of fog rolling in over a mountain range present difficulties. There is also the case of a distant horizon seen over a nearby crest: the near skyline may be the one of operational importance.

We have chosen a range of images for testing. *Portland* shows a city skyline against a cloudy sky. *Mountain* is a distant mountain against a nearly clear sky. *Bishop* contains a near skyline and a distant one that merges with a cloudy sky; it is difficult for untrained observers to segment. All of these images were reduced to 128x128 to save execution time and to permit use of the rundisplay option.

An early test with the *portland* image at full 512x512 resolution was disappointing. It was done with red, green, and blue input feature planes and with the original default threshold settings. (In particular, **hsmooth** was 1 and **height** was 70. **Maxmin** was also set to 100 in order to get the segmentation started.) The resulting segmentation was erratic, locating many small details while missing several obvious regions. In particular one white building was not distinguished from the blue sky even though most of its windows were found. Numerous tiny patches of sky were segmented out for no apparent reason, yet an easily visible U.S. flag was not distinguished from the sky region.

Subsequent analysis and experimentation led to several improvements: minor software bugs were fixed, the strict/moderate/mild parameter scale was developed, and Kender's versions of the HSD and YIQ transforms were implemented.[1] The D and Y transforms are essentially redundant, and are also very similar to the red, green, and blue feature planes. They do not always segment identically, but the extra

[1]Subsequent correspondence with Steven Shafer indicates that CMU researchers have favored Ohta's transforms over the nonlinear HSD scale.

information is not worth the computational effort. We have used all nine features, however.

Hue was mapped to the range 0 to 179, with red at 0 (and 180), green at 60, and blue at 120. Achromatic pixels (*i.e.*, black, gray, and white) were mapped to 255; this ultimately made no difference since pixels with exactly equal red, green, and blue components are exceedingly rare. A less exact test for achromaticity might work better (or at least differently) for images with slight imbalances in the color strengths; the *bishop* image, for instance, is found to have red clouds even though they appear white.

Pixels containing blue mixed with red (*i.e.*, purples and violets) are also rare even in the hazy mountain scenes, so we found no particular problem with peaks in the hue histogram being split between the bottom and top portions of the scale. Saturation was more likely to have such instabilities; we found examples of dark or shadowed image regions that transformed to very high saturation values. The histograms resembled peaks with their left tails clipped at zero and moved up to the high end of the scale. Such areas were so small in our test imagery that they never caused any difficulties.

The I and Q color features computed by Kender's formulas must be divided by two (and then shifted to a nonnegative range) if they are to be stored in 8-bit image planes. Compression to eight bits is not really required by PHOENIX, but it seems a reasonable dynamic range. Experiments showed, however, that most of this range was being wasted. We chose to stretch I by a factor of two and Q by a factor of four prior to quantization, with clipping of extreme values. This greatly increased their usefulness for natural imagery, although it could fail for scenes containing large regions of saturated colors.

For skyline delineation, hue was the most important feature. Sky, clouds, and some vegetation all had hue values near blue or blue-green, whereas land and buildings were closer to red, orange/brown, and yellow. This might not hold true for other scenes, but, for our *portland* and *mountain* images, the hue feature and the strict parameter settings were nearly sufficient to extract the sky as a single region. For the *bishop* image they extracted the near skyline from the rather homogeneous background of distant land and cloudy sky.

Even better results were obtained by first segmenting with strict heuristics and then resegmenting with moderate heuristics. (This involves somewhat more computation than using the moderate heuristics alone, but did a better job of segmenting textured regions.) The strict heuristics typically produce three to five regions for a 128x128 image, and the moderate heuristics extend this to 12 to 30 regions. Further segmentation with the mild heuristics produces 60 to 100 regions, many of them shadows or contours in fairly smooth scene regions. Some of these contours may be due to instabilities in the color transforms, but most have visible interpretations.

Skyline determination was straightforward in the *portland* and *mountain* images because the sky was extracted as a single region. The *bishop* image was much more difficult. Strict and moderate heuristics separate the nearby land, blue sky, several large cloud areas, and a large region that included a distant valley, a rim of mountains, and a cloudy sky. The true skyline could only be segmented by using the mild heuristics. It was found as a single boundary, but could easily have been broken apart if the various thresholds had been slightly different. In any case the challenging problem of determining which regions were sky and which were land is not resolved by PHOENIX; it just passes the regions on to some unknown post-processor

(in this case a human visual system).

The *bishop* image exhibited another characteristic of PHOENIX. In segmenting the blue sky from large cloud masses, it misplaces the boundary slightly. This is because the histogram cutpoints are sensitive to global area effects rather than local spatial variations. (Shafer [Shafer80, Shafer82] discussed this as the "majority rule" problem.) The misclassified cloud patches are picked up during later segmentations, but are so small that many are remerged with the sky. PHOENIX currently has no way of detecting the spatial patterns of small noise patches that indicate a poorly chosen border or a string of mixed-source pixels.

A final test sequence was run on the full-resolution (500x500) *portland* image. Strict and even moderate heuristics were unable to segment the image when only the red, green, and blue feature planes were used; it was necessary to use the mild heuristics. The best approach would be to start the segmentation with mild thresholds and then return to strict or moderate ones for segmenting the subregions. Instead, we avoided such special interference and ran the segmentation to completion using mild heuristics. The full run (which, with the 'v' flag set, generated 19,000 lines of printout) required 33 minutes of CPU time:

PHASE	REAL	CPU
Fetch	0:00:13	0:00:08
Histogram	0:04:13	0:02:32
Interval	0:18:12	0:07:27
Goodfeatures	0:00:01	0:00:00
Nextfeature	0:00:01	0:00:01
Threshold	0:10:00	0:03:47
Patch	0:03:51	0:03:30
Evaluate	0:00:05	0:00:04
Selection	0:00:06	0:00:05
Collect	0:38:12	0:14:04
Segmentation	1:18:15	0:32:34

The final segmentation into 1182 regions (including nearly every window of every building) was much better than the original attempt, but still had difficulties distinguishing a glass-surfaced building from the sky that it reflected. The U.S. flag was segmented out as two small regions.

Another attempt was made using color transforms. This time the strict heuristics were able to segment sky from land using the hue feature. Results were very similar to those for the reduced *portland* image, although outlines were noisy and somewhat more "gerrymandered." Splitting on the hue features required less than two minutes of CPU (with perhaps an equal amount for computing the color transforms) and produced nine regions, one of which was the U.S. flag. Some vegetation and building surfaces were included in the sky region, including most of the glass-surfaced building. It took another minute to determine that the nine regions were homogeneous.

Further segmentation required switching to the moderate heuristics. Running this sequence to completion produced 153 regions after an additional 11 minutes. The sky was cleanly separated from the vegetation and buildings, but had been split into two major regions along a front in the cloud cover. The noise threshold of 10 was evidently too low for this task and image resolution, but only a few small regions were

retained. This combined strict/moderate segmentation of the color transforms was very successful at skyline delineation. (Segmentation using the moderate heuristics alone is also quite good.)

The regions found by PHOENIX are not smoothed in any way. Often they are narrow, twisted, or convoluted. This contrasts with human segmentation, which favors straight lines at the expense of region homogeneity. Despite this, the regions found with the strict and moderate thresholds are quite reasonable, and even the mild thresholds give acceptable segmentations. The best course seems to be to oversegment the image and then use some type of post-analysis to classify and merge the regions.

For the particular application of skyline delineation, PHOENIX is handicapped by its lack of knowledge about the task domain. It spends much of its time segmenting and resegmenting areas that are nowhere near the skyline. A more focused search would save computation and pass fewer regions for further analysis. Specific feature planes for land/sky segmentation might also be used to simplify the segmentation and classification task.

6.3. Summary

PHOENIX is a general-purpose segmentation system. It is designed to produce a reasonable segmentation on almost any type of imagery. Proper use of the system requires extensive knowledge of the algorithm and of the effects of various threshold settings, but the system can be made to produce reasonable segmentations.

A difficult part of the Testbed integration effort was the analysis and documentation of PHOENIX control options and heuristic thresholds. Eventually this work led to the strict/moderate/mild threshold settings specified above. The various settings were determined by analysis, by disabling most heuristics and testing the remainder in isolation, by watching the heuristics interact during segmentation of a simple *chair* image, and by refinement during segmentation of natural imagery. While possibly not optimal for any particular purpose, these threshold groupings provide a framework for fine adjustments.

Evaluation of segmentation software is a difficult task. There are few methods for comparing segmentations other than tabulation of pixel classification errors [Yasnoff77] or subjective evaluation on simulated or natural imagery [Nagin79, Ranade80]. We have subjectively evaluated PHOENIX's performance for a particular task using a variety of images.

PHOENIX performed adequately for the task of skyline delineation. We did not develop optimum parameters or procedures for this task, but used very general techniques developed for much simpler test imagery. The amount of computation that PHOENIX required to find the skyline varied with the difficulty of the scene, but it did succeed in all cases. The further problem of determining which regions constitute sky is beyond the domain of this system.

Section 7

Suggested Improvements

The process of evaluation has turned up numerous ways to improve the current PHOENIX implementation. Comments about existing features have been made at the appropriate points throughout this document. The following are additional suggestions for substantial modifications or needed research. Some of these would require major research projects or are beyond the scope of a segmentation program *per se*. (The large number of suggestions should not be taken as a criticism of the PHOENIX system. Rather it is a tribute that the approach is flexible enough to support such extensions and is promising enough to be worth the effort.)

- *Flexible Interaction*

  PHOENIX is both an automated segmentation system and an interactive one. The interactive control system is excellent, but could be improved if more of the dynamic decisions were based on queues or lists instead of compiled iterations. The user could then attach and detach feature planes, manually screen or add histogram cutpoints, select heuristics to be applied; accept or reject thresholded patches, *etc.*

- *Alternate Color Features*

  Our experience indicates that the hue feature is much more useful for skyline delineation than the original color features. Researchers at CMU have favored Ohta's transforms over HSD features for general work. LANDSAT analysts have used various ratios of color bands to emphasize water, vegetation, mineral deposits, *etc.* There may still be much to be gained by developing transforms suited to particular tasks.

- *Texture Transforms*

  Texture features supplied to PHOENIX evidently need to be combined in much the same way that color features are combined into YIQ and HSD versions. Combinations of texture and color features might also be useful for splitting the one-dimensional projections of multidimensional histogram peaks.

- *Additional Feature Types*

  PHOENIX has been developed primarily for color image segmentation, although it seems able to work in other multispectral and multitextural domains. Since additional features can only improve performance (at a cost in processing time), it may be desirable to add other computed scene

characteristics such as gradient and edge maps; stereo disparity; estimated illumination at each pixel; estimated surface distance, reflectance, curvature, and orientation [Horn77, Barrow81, Brady82]; optic flow [Thompson80]; and estimated material type.

* *Delayed Transforms*

PHOENIX currently accepts color transform features (YIQ, HSD, *etc.*) as input feature planes. This works well in a research environment, but might require more storage and computation than necessary for a production environment. These feature planes and histograms can be computed from the image as needed. Perhaps few regions would require thresholding on transformed values if RGB segmentation were first used wherever effective.

* *Histogram Stretching*

Some of the color transform features are likely to have a narrow range on any given image, making them useless for segmentation. Unfortunately the feature ranges vary from one image to another. Since PHOENIX is not sensitive to linear transformation of the features, it might be wise to stretch each feature to its full dynamic range prior to quantization. (This requires an initial pass through the image to determine the range.) This computation could even be done on a region by region basis. Note, however, that full nonlinear histogram equalization will prevent PHOENIX from segmenting the feature at all.

* *Adaptive Smoothing*

PHOENIX currently applies the same smoothing window to each of the feature histograms. An adaptive or iterative smoothing algorithm that suppressed noise without merging peaks would perform better.

* *Luminance Screening*

Ohlander and Price [Ohlander78] segment first on high and low luminance (Y or D) values to avoid singularities in the color transforms. PHOENIX counts on spatial analysis to reject these unstable transform intervals, but might benefit from similarly extracting bright and dark regions before doing more general segmentation[1]. An alternative is to change the color transform code so that colors in the unstable regions are all mapped to special code values; PHOENIX might then need to understand these mappings.

* *Histogram Modeling*

Several comments were made in Section 6 about the deficiencies of PHOENIX's interval selection algorithm. The most severe problems relate to

[1]This capability is now available in the CMU version of PHOENIX.

its lack of a model for histogram peaks or valleys. Although its heuristics are cheap and often effective, there may be better alternatives. Statistical modeling has already been mentioned. Spline fitting, Kalman filtering, filtered gradient zero-crossing detection, and hierarchical waveform parsing [Ehrich76] are others. Another idea is to use one set of heuristics to assign a "valley center" score to each histogram bin and another set to select high-scoring bins that are spaced suitably far apart.

* *Circular Features*

Hue is computed on a circular interval, with red at both ends of the scale. The histogram analysis routines could be modified to understand this characteristic so that purple/red peaks would not be eliminated or split. The PLAN segmenter [Price76, Ohlander78] has this capability.

* *Feature Rejection*

A feature may fail to segment a region either because it contains broad peaks that cannot be resolved or because the histogram has degenerated to a narrow spike. Although the latter is not too common, some computation could be saved by eliminating such a feature from all further splitting of the region and its subregions.

* *Reordered Heuristics*

Questionable heuristics such as **relarea**, **height**, and **absmin** should be postponed as long as possible in order to develop context and perhaps eliminate the need for the decision. A supervisory system might be added to determine when these tests are required, and multiple spatial analyses might be performed as a final check.

* *Alternate Heuristics*

The absolute heuristics (*e.g.*, **absarea** and **noise**) can only be set in the context of a particular task and image resolution. Relative heuristics are generally better, although the PHOENIX versions often perform differently for small regions than for large ones. Also good are those, like **intsmax**, that rank order the histogram cutpoints and choose the top few.

PHOENIX and the Ohlander/Price segmenters use slightly different heuristics for segmenting histograms. In particular, Price's version prefers bimodal features and also considers the heights and slopes of neighboring peaks (to avoid chopping off the tail of a skewed peak). There is also a special heuristic for extracting a low-saturation interval. Such heuristics could easily be added to PHOENIX, although it is not clear how they would interacted with the existing heuristics.

Once histogram peaks have been found, Ohlander and Price use successively weaker acceptance criteria to choose a single histogram peak for thresholding. This differs from the PHOENIX approach, which uses successively

stronger rejection criteria to screen potential cutpoints. While either set of heuristics might be transformed to the other system, it is not clear how acceptance and rejection criteria could be made to work together.

A useful property of PHOENIX's heuristics is monotonicity; once used, later applications of the same heuristic would have no effect. If other heuristics were introduced that destroyed this property, it might be necessary to repeat the heuristics round-robin until the entire set produced no change in the cutpoints.

Perhaps the best advice is to make the heuristics so intelligent that each individually seldom makes a mistake, and to make them accessible to the user so that they can be refined when exceptions are found. This is the expert systems approach, with part of each heuristic being a test to determine when the rule is applicable.

* *Multivariate Histogram Analysis*

Clustering and multivariate histogram segmentation are discussed in Appendix A.6. There may be situations in which a single three-dimensional histogram analysis is more powerful and less expensive than PHOENIX's sequential univariate analyses of (typically) nine histograms. Histogram storage and analysis are becoming much less of a problem as computer hardware improves, and a single clear-cut decision in multidimensional space may often take the place of many doubtful decisions in the one-dimensional spaces.

* *Adaptive Cluster Analysis*

Most clusters in a multidimensional histogram space can be adequately separated by piecewise-linear decision boundaries. These decision surfaces can be found by standard cluster analysis techniques without storing multidimensional histograms. The advantages increase as the number of features considered increases, since the adaptive cluster methods require essentially the same analysis time regardless of dimensionality. There are additional advantages to using parametric (*e.g.*, Gaussian) methods when appropriate, since they are designed to optimally separate peaks from each other and from random noise.

* *Conservative Thresholding*

The region boundaries computed with PHOENIX are affected by global circumstances such as the number and size of other similar regions. An editing phase may correct the boundaries by moving them back and forth and by deleting noise regions, but will not be able to recover small regions that have been absorbed by their neighbors. The occurrence of such lost regions can be minimized by conservative thresholding [Nagin77]. Some type of region growing is then needed to merge pixels between regions. One method of adjusting region boundaries is given in [Barrett81].

* *Noise Analysis*

PHOENIX currently discards any region that is too small either in absolute area or as a percentage of its parent region. Even for task-independent segmentation this may be too simple; any meaningful interpretation of some small patches would sharpen the **retain** test based on the remaining noise. There may also be applications for which the small anomalous patches are important and cannot be discarded.

In the most common situation, poorly segmented or mixed-source pixels are discovered along a region boundary. PHOENIX remerges these with the parent region instead of testing to see which region should properly contain them. (This could be done by disabling the noise heuristics and allowing a post-processor to make such decisions, but, with the noise heuristic disabled, PHOENIX has no way to choose which feature to use.)

A more difficult case arises for occluded objects or "flocks" of related pixels. The disconnected parts have similar histograms and are located by the histogram analysis, but spatial analysis rejects the feature or merges the patches. This is right for most applications, but wrong for others. A more sophisticated system would analyze the small patches for shape, contrast, regular spacing, similarity to existing regions, multispectral signature, or other unifying criteria.

* *Planning*

PHOENIX does not currently include the planning mechanisms developed by Price [Price76]. These would seem worth inclusion in either a research or a production system. The software involved is similar to that for conservative thresholding.

* *Partitioning*

Another neglected feature of Price's system is partitioning of large regions. Price uses thresholds derived from the subregions to segment the entire scene — this gets the segmenter started when faced with unimodal histograms. An alternative is to analyze each subimage independently, then merge the region descriptions in a later editing step. The method performs badly if the arbitrary divisions are close to true region boundaries. While this can lead to some blockiness, it reduces computation time at a very small sacrifice in global information.

* *Selective Sampling*

The problem of finding small regions within large ones may also be combatted by computing histograms only near pixels with high gradient [Weszka74]. Equally valid is the use of only low-gradient pixels; this resolves the centers of large regions but may produce poor boundaries. Such techniques could be used after recursive segmentation of a region can proceed no further. They are made easier if a gradient map is one of the input features.

Suggested Improvements

* *Relaxation Analysis*

  Often a histogram is obviously bimodal, but the peaks cannot be resolved. PHOENIX allows the feature to be used for splitting, but may not be sophisticated enough to merge the resulting noise regions into a meaningful segmentation. For such cases, or even for unimodal regions, more expensive analysis may be appropriate. Use of more features, partitioning, and selective sampling have already been discussed. If all else fails, one can modify the original image by nonlinear relaxation to smooth the subregion interiors and enhance the boundaries [Bhanu82].

* *Map Input*

  One method of adding planning and feedback is to feed crude segmentation maps to PHOENIX as feature planes. These maps might come from previous PHOENIX runs or from other segmenters. Using such maps requires different control structures and heuristics since the bin contents, not the overall histogram peaks and valleys, are the meaningful features. Phoenix can make partial use of such a segmentation map only by accepting it as the current state and then trying to split it further. A more flexible system might use multiple segmentation maps as guides to a multidimensional cluster analysis.

* *Adaptive Thresholds*

  The Ohlander and Price segmenters use a tightly constrained valley selection heuristic, then a weaker one. A similar interactive technique has been found useful with PHOENIX. This concept could be integrated with the PHOENIX control structure by automatically segmenting first with severe histogram smoothing and tight constraints, then with gradually relaxed constraints for regions that are deemed worthy of further effort. Each new region would go through this same sequence of tests. The cost of such a technique would be lessened if the pre-smoothed histogram were retained until a satisfactory segmentation was achieved.

* *Shape Analysis*

  PHOENIX currently chooses a region for segmentation without regard to the region's shape or context. Only the region size and segmentation depth are considered. It is possible that better segmentation could be achieved by considering shape during the ***fetch*** phase and also during spatial analysis. Extended regions such as rivers and roads may require heuristics different from those for compact regions.

* *Heuristic Training*

  The space of all heuristic orderings and threshold settings is too large for intuitive design. If the heuristics are to be extended or improved, some type of ordered search is required. This will require a set of training images with known region boundaries. PHOENIX can be modified so that segmentation

errors are flagged and evaluated at each step. A human operator or higher-level control system could then drive PHOENIX through the training set, adjusting the thresholds to achieve good performance. If ground-truth training images are not available, a much more sophisticated expert system will be required.

* *Additional Displays*

The rundisplay option is very good, and made it much easier to evaluate the existing heuristics. The SRI heuristic display (flag H) should be extended to show separately the action of the **absarea** and **relarea** heuristics, and of **absscore** and **relscore.** (It should also be modified to allow early escape from the full set of displays. The best solution would be to make each heuristic application a separate phase.)

The rundisplay layout of all feature histograms on a single screen is also excellent, although it could be improved by printing the interval set score with each histogram. A similar display should be implemented for the "display histograms" command, which currently shows the histograms one by one. For single-feature interval set display, each interval set area should be printed; a vertical scale on the histogram might also help.

For large images, where rundisplay is currently not available, it would be useful to be able to display the histograms of any region at any time. At present this usually involves moving the region to the segmentation queue and executing a *histogram* phase. This cannot be done if the region has already been segmented unless you are willing to prune the region.

Better displays are also needed for showing individual regions in context. This is currently done by drawing the region outline on the original image and marking the center with a blinking cursor. Unfortunately the outline is often difficult to see and the cursor is insufficient to indicate whether the inside or outside of the outline is meant. A better display would show either the region or its surround as a solid patch. (A keystroke could be used to flip between the two options.)

During rundisplay each region that is created is drawn as an outline on the original image. This overlay is supposed to represent the current state of the segmentation. It should be erased and redrawn when a region is pruned. Some of the other rundisplay components should be erased when a retry command makes them obsolete.

* *Immediate Feedback*

The noise area produced in a *threshold* phase is not reported until after all promising features have been analyzed. It would be better to report the results of the spatial analyses individually as well as jointly; the user could then match the noise statistic with the corresponding patch display. (The *evaluate* phase does little except compute and print these percentages. It could be eliminated.) Another improvement would be to inform the user about which heuristic rejected a particular interval set score.

Suggested Improvements

* *Verbosity Coordination*

  The PHOENIX code contains several mechanisms for controlling verbose printout and debugging messages. Various messages are controlled by compiler flags, global variables, PHOENIX flags, and by the SRI **printerr** package. It would be better if all were controlled by PHOENIX flags or variables. There should be an additional flag to print the name of each phase as it is begun; this would simplify debugging and retry commands.

* *Queue Management*

  PHOENIX maintains a segmentation queue and a terminal region queue. It is somewhat disconcerting when the same region appears on both, or when a region appears several times on one queue. PHOENIX does check each fetched region to make sure that it has not been segmented, but a better approach would be to ensure that the queues remain valid at all times. Adding a region to a queue should remove all other occurrences, and segmented regions should not be allowed on the segmentation queue. The queue manipulation routines should also be augmented with various screening options for transferring regions from one queue to the other.

* *Split/Merge Capability*

  One option that the user should have is to combine two neighboring regions. Eventually heuristics might be added for doing this automatically in appropriate circumstances. The segmentation history will require a general graph representation instead of a tree.

* *Explanatory Capability*

  It would also be helpful if enough history information were kept so that the system could answer questions about the final segmentation and the steps that led to it. This would include questions about why a particular region had been retained and why it had not been split further, what thresholds would be needed to segment it further, what effect those thresholds would have on other regions, *etc.* (Admittedly some of the answers might require extensive computation.) Such question-answering capabilities are common in expert systems. The answer to a "why did you" question is typically a printout of the rule that triggered the action.

* *Coroutine Implementation*

  PHOENIX can be driven by another program, but the interaction is clumsy. The driver program must invoke PHOENIX and send commands down a UNIX pipe. Output is obtained by sending a "checkpoint" command and then examining the resulting map and data file.[2]

---

[2]This solution was suggested by Steven Shafer at CMU. It avoids the checkpoint parsing overhead of repeatedly invoking new PHOENIX processes with the single-step option.

For more flexible interaction, PHOENIX must either be implemented as a subroutine or as a server process. The subroutine approach gives the control program a dedicated process for segmenting a particular image; some communication protocol would be needed for conveying the new segmentation results. The server, or coroutine, implementation is more like having a separate piece of hardware for segmenting images: the control program would send requests, and PHOENIX would send back replies. This permits isolation of the PHOENIX history files so that no other program would have to load and parse them, but it does introduce complications if PHOENIX services are to be shared by several control programs.

Section 8

Conclusions

The PHOENIX segmentation system is one of several existing systems for recursively segmenting digital images. Its major contributions are the optional use of multiple thresholds, spatial analysis for choosing between good features, and a sophisticated control interface. Some of the strengths and weaknesses of the PHOENIX algorithm are listed below.

* PHOENIX, like other region-based methods, always yields closed region boundaries. This is not true of edge-based feature extraction methods, with the possible exception of boundary following and zero-crossing detection [see Appendix A]. Closed boundaries are the essence of segmentation and greatly simplify certain classification and mensuration tasks.

* PHOENIX is a hierarchical or recursive segmenter, which means that even a partial segmentation may be useful. This can save a great deal of computation if efforts are concentrated on those regions where further segmentation is critical. If PHOENIX is to be driven to its limits, other methods of segmenting to small, homogeneous regions may be more economical.

* PHOENIX is relatively insensitive to noise. Thresholds are determined by the feature histograms, where noise tends to average out. This contrasts with edge-based methods, where the local image characteristics can be highly perturbed by noise.

* Different segmentation problems require different amounts of histogram smoothing [Ranade80]. It generally works best to start PHOENIX with strong smoothing and strict heuristics and then to gradually weaken both. Some images, however, require mild smoothing or thresholds to get the segmentation started. An adaptive system would be desirable.

* PHOENIX has no notion of boundary straightness or smoothness. This may be good or bad depending on the scene characteristics and the analysis task. It easily extracts large homogeneous regions that may be adjacent to detailed, irregular regions (*e.g.*, lakes adjacent to dock areas or sky above a city); such tasks can be difficult for edge-based segmenters.

* PHOENIX tends to miss small regions within large ones because they contribute so little to the composite histogram. It is thus poorly suited for detecting vehicles and small buildings in aerial scenes, although there may be ways to adapt it to this use. It also tends to misplace the boundary between a large region and a small one, thus obscuring roads, rivers, and other thin regions. Boundaries found by edge-based methods are less affected by distant scene properties.

- PHOENIX may also fail to detect even long and highly-visible boundaries between two similar regions if the region textures cause their histograms to overlap. Edge-based methods are better able to detect local variations at the boundary.

- PHOENIX requires multispectral or "multitextural" input for effective operation, and may even require transformations and combinations of these feature planes. Edge-based techniques are better adapted to operation in a single feature plane.

- Since perfect segmentation is undefined and unobtainable, PHOENIX must oversegment an image in order to find all region boundaries that may be of use to any higher-level process. It is left for a segmentation editing step to merge segments that have no usefulness for some particular purpose. Without having such a step, or indeed even a purpose, it is very difficult to evaluate the segmenter output.

Selection of a segmentation algorithm and improvement of a particular software package are both highly dependent on the task to be performed. The PHOENIX segmentation system is a flexible starting point for further development. This report and the SRI Testbed environment help to make PHOENIX available as a benchmark system and as a research tool.

Appendix A

Alternate Segmentation Techniques

This appendix explores alternate methods of segmenting images. It is intended to clarify the issues involved in region extraction, and to introduce background and vocabulary needed to read the literature in this field. For other surveys see [Zucker76a], [Riseman77], and [Fu81].

A.1. Edge Methods

One approach to segmentation consists of detecting small edge elements and then linking them into region boundaries. Edge and region methods are nearly equivalent for simple scenes of cubes and wedges. In natural images, specific structures are best found with particular techniques [Nevatia77a]. Region methods locate irregular, homogeneous regions, but may ignore or conceal linear features; edge methods detect linear features and detailed (or possibly camouflaged) objects, but give fragmented region boundaries that may be difficult to interpret. Perhaps the two must be combined so that detected edges provide context for region growing and region knowledge can aid edge linking [Milgram77, Milgram78, Barrow81].

Sometimes edge detection and linking are combined [Pingle71, Montanari71, Martelli76]; this is called edge following or boundary tracking, and has advantages when closed regions are required. A similar method is run tracking [Nahi77, Nahi78], in which the object boundaries found on one row are used to aid location of boundaries on the next row. (This is similar to the PHOENIX connected-component extraction algorithm.)

A separate edge detection step is more popular because it is compatible with either single-pass or parallel implementation, and because the detected edge elements are also useful as texture primitives. Edge linking may be done using relaxation labeling [Riseman77, Zucker77, Prager80], expansion-contraction to close gaps [Perkins80], curve fitting, or clustering and heuristic linking [Jarvis75, Nevatia76, Fischler83].

Edges in digital images are difficult to define. A few edge detectors are based on theoretical models of scene edges [Hueckel71, Hueckel73, Horn77, Mitiche80, Haralick81, Brady82], but most are heuristic local gradient estimators [Davis75, Pratt78]. Some operators are small in order to approach a true local derivative, others are quite large to provide noise immunity. Comparative studies [Fram75, Bullock76, Abdou79] have not proven the superiority of any one operator for all classes of imagery.

Color edges are even more difficult to define. Either a single gradient map must be defined on the multivariate feature plane, or edges detected separately in each feature plane must somehow be combined [Nevatia77b, Robinson77]. For color data the method should match human perception of color edges, but we would like it to extend to texture features and other data as well.

Texture edges (*i.e.*, boundaries between regions of differing texture) are also important. The standard approach is to identify ordinary intensity edges in some texture transform of the image, but texture-specific methods have been developed [Thompson77, Deguchi78, Davis80, Davis82].

Some exciting advances have been made in the area of zero-crossing detection [Grimson80, Brady82]. The image is convolved with the second derivative of a Gaussian blur function (chosen to match hypothesized channels in the human visual system). Zero-crossings in the filtered image then form closed region boundaries whose positions can be estimated with sub-pixel accuracy [MacVicar-Whelan81]. Further, the sensitivity of the detector to edges of different widths can be controlled by the width of the Gaussian function, and the strength of the edge at a given point can be measured by the rate of change across the zero crossing. More work is needed to determine how to combine these multiple sources of evidence without losing the closed-region property.

A.2. Thresholding

Thresholding is a quick way of locating regions. Often an image function may be found that is maximal for the smooth interiors of regions and minimal for region boundaries. Other functions, such as the image itself, may be maximal in some region centers and minimal in others; boundary areas take on intermediate values. In either case, thresholding may be used to separate region interiors from edges.

Using successively lower thresholds generates a contour map; adding a stopping criterion makes this a segmentation algorithm. In forward-looking infrared (FLIR) target imagery it has been found that object shapes change very little as the threshold is varied, but noise regions change dramatically. Milgram [Milgram77] exploits this consistency by choosing the threshold giving the best match between corresponding region boundaries and the edge elements detected by another method; this has difficulties with small or textured regions [Ranade80].

There are three types of threshold: *constant*, *scene-dependent*, and *adaptive*. ([Weszka78] further classified thresholds as *global* if they depend only on pixel value, *local* if they depend on neighboring pixel values, and *dynamic* if they depend on spatial position.)

Constant thresholds are those having the same value for all images (*e.g.*, [Kasvand74]). Some real-time hardware systems use this technique, but it is rare for any function of diverse images to have an appropriate constant threshold.

Scene-dependent thresholds are constant for a given image, but may vary as a function of the sensor, illumination, analysis task, or image content. The threshold is typically set interactively by an observer or automatically by histogram analysis. Histogram thresholding was developed in the context of cell segmentation and identification [Prewitt66], and may still be the best technique for this purpose [Ranade80].

Relaxation processes have also been used to remap the histogram into a few dominant intensities [Rosenfeld78, Peleg78, Ranade80]; this is essentially a thresholding process. Like other histogram-based methods, it tends to ignore small regions that may be semantically meaningful.

Adaptive thresholds are set automatically as a function of local scene content; they

vary from point to point within an image. Such thresholds can adjust for changes in illumination within a scene. As usually implemented, the threshold is a function of the image data along a scan line [Serreyn78] or within a window. The threshold will work badly if a window contains no object or multiple objects with different intensities. An isolated small object may be overlooked in a large window, and a large object may be thresholded inconsistently across small windows.

Histograms computed over regions of mixed sizes are difficult to segment. Weszka *et al.* [Weszka74] suggest computing the histogram only over pixels near region boundaries (*i.e.*, pixels with high gradient). Further discussions of edge detection and texture analysis to set thresholds may be found in [Weszka78] and [Kohler81].

Panda and Rosenfeld [Panda78] found that intensity/edge-strength histograms of FLIR targets are trimodal, with peaks representing background, edge, and object. It was found insufficient to set a single threshold at the intensity value of the edge peak. Better methods used edge gradient to implement decision boundaries extending from the edge peak to the valley between the background and object peaks.

A.3. Iterative Modification

An alternative to adaptive thresholding is context-sensitive modification of the image itself. This is typically done by iterative relaxation or "competitive-cooperative" processes [Troy73, Hummel78, Zucker78, Kirby79, Nagin79, Eklundh80, Peleg80], although single-pass methods such as cluster analysis and pixel classification could be adapted to this purpose. (Relaxation output might be useful in training such a classifier.)

Unfortunately relaxation processes tend either to do very little or to be very sensitive to the updating rule, the image-dependent compatibility coefficients, or the class membership function for initially labeling each pixel. Various schemes have been proposed for estimating these quantities. Histogram segmentation, for instance, can be used to select the initial class membership function [Ranade80].

One use of relaxation is to get the segmenter started on scenes (or composite regions) with unimodal histograms [Bhanu82]. The relaxation process emphasizes spatial features that are too weak or space-variant to show up in the histogram. Such preprocessing can split a composite peak into subpeaks that are useful to a threshold segmenter. This is in contrast to relaxation methods applied to the histogram (see Section A.2), which can reduce the number of peaks but never create new ones.

A.4. Recursive Splitting

Uniform regions can be found by recursively splitting nonuniform regions (beginning with the whole image) into smaller regions. In the limit this produces single-valued and perhaps single-pixel regions. In some cases it may be desirable to split even uniform regions using region shape criteria [Lemkin79, Rutkowski81].

Since almost any area can be better represented (in a mean-square-error sense) by two small regions than by a single large one, it is difficult to determine when to stop splitting. Most splitting methods lack a justifiable stopping criterion. One possibility, derived from coding and information theory, is to use the number of bits required to

code a region before and after splitting as a measure of improvement; this is unfortunately dependent on the coding method.

Splitting is always costly since region descriptors (shape, variance, *etc.*) must be computed for each subregion. Suppose that a region is split into d subregions: all pixels in at least $d-1$ subregions must be reexamined to compute the new descriptors. To segment an image down to the pixel level requires

$$N^2 \left(1+\frac{d-1}{d}\log_d N^2\right)$$

pixel examinations, as opposed to N^2 for segmentation by merging or growing procedures.

The above analysis assumes a deterministic splitting algorithm. In quadrant subdivision, for example, regions are repeatedly split into four square subregions until homogeneous regions are found. (The number of pixels in a row or column is typically a power of two, making the subdivision trivial.) This method segments too finely so that a later merging step is required; even so, it is one of the fastest partitioning methods.

The most difficult step in other partitioning methods is deciding exactly where the new boundary should go. If the new boundary location is not known *a priori*, the region descriptors must be computed for each possible boundary. This can involve a very large search space and enormous computational costs. Functional approximation schemes [Pavlidis72] avoid this by using parametric solutions for the boundary and for the region descriptors. The PHOENIX algorithm offers another solution by choosing boundaries along significant intensity contours.

A.5. Classification

The purpose of segmentation is often classification. This can be reversed by using pixel classification to achieve segmentation. The basic problem is to classify an image window as one of several texture types. For a survey of multispectral classification in remote sensing see Haralick [Haralick76].

The method of maximum likelihood could be used if we had enough information about the texture classes. We would estimate the likelihood of the observed pattern under each hypothesis, then choose the texture class giving the highest likelihood. Unfortunately the required probability distributions are too large to be represented as histograms.

Nonparametric methods have been proposed for estimating and storing large distributions; see, for example, the set covering procedures of Read and Jayaramamurthy [Read72] and McCormick and Jayaramamurthy [McCormick75]. It seems sensible, however, to assume a parametric form for the distributions whenever it is possible to do so. This allows us to develop simple vector product scores for classifying pixels.

Image intensities seem to be well characterized by statistical moments. Ahuja *et al.* [Ahuja77] show that the first few moments are as useful as an entire histogram for classifying textures. Statistical methods have also been developed for classifying the spatial distributions of texture pixels [Haralick73, Mitchell78, Rosenfeld79, Laws80].

A simple nonparametric approach is to store an exemplar (or feature vector) for each known texture type. Each pixel to be classified is compared to each exemplar

and is assigned to the class of the most similar one. This has the advantage that it is easy to add additional texture exemplars.

The principal difficulty with any type of texture classification is that the region to compute texture statistics over cannot be known unless segmentation has already been accomplished. Typical image processing problems require analyses near the resolution limit of the imagery, and windowing errors are intolerable.

A.6. Clustering

Cluster analysis is identical to classification except that the classes are not known *a priori*. Various spectral or spatial descriptors of the pixels are analyzed for similarities, and those pixels that are judged similar to each other or to some prototypical seed pixels are assigned to the same class [Wacker69, Carlton77, Goldberg78, Yoo78, Coleman79, Mitchell79, Schachter79]. Spatial analysis then completes the segmentation; this analysis may include probabilistic relaxation [Nagin79, Kohler81] or other methods of noise cleaning and boundary smoothing. Clustering can also be used to merge regions found by thresholding or other methods [Haralick75a].

PHOENIX-style histogram segmentation is a type of cluster analysis. This is more evident when done in a multivariate space [Schachter75, Schachter77, Hanson78, Milgram79, Schachter79, Milgram80]. Multivariate histograms are typically quantized very coarsely in each feature in order to reduce storage requirements and analysis time. If finer quantization is required, either two passes should be made through the data (planning), or an adaptive accumulator scheme should be used [Schachter75, O'Roarke81, Sloan81]. Perhaps a better alternative is to use a parametric or adaptive (*perceptron*) cluster method not relying on histograms.

A.7. Region Growing

Region growing is based on the premise that it is easier to identify interior pixels than border pixels. One starts with a set of region seeds, preferably one seed per image region. Each region is then expanded like a wavefront, incorporating adjacent unassigned pixels. Growth stops when all pixels have been absorbed or when unassigned pixels are too dissimilar to be merged with adjacent regions. An editing phase may follow in which unassigned pixels are classified and neighboring regions are tested to see if they can be merged.

One method is to start with completely homogeneous regions and then merge neighbors that have statistically-similar pixel populations or classifications [Muerle68, Gupta74]. Another is to merge neighbors that are divided by "weak" boundaries or that together form a simple shape (the "phagocyte" heuristic) [Brice70]. Yet another is to accept any unassigned pixel as a region seed and to grow the region until its natural limits are found. The regions may be grown either sequentially [Jarvis75] or in parallel during a single scan [Yakimovsky76]. Any of these methods essentially combine connected-component extraction with region growing.

Region seeds are usually found by crude segmentation, retaining as seeds only those pixel groups most certain to belong together. The seeds may be chosen interactively [Garvey76b] or automatically. Often the seeds are chosen by adaptive thresholding or peak-finding algorithms applied to a gradient or edge transformation of the image. The segmentation is then done on the original image data. (Region growing typically

uses only monochrome input, although see Kettig [Kettig76].)

Levine and Leemet [Levine76] have developed an interesting method of obtaining region seeds from an edge map. The edges are thickened by pyramid reduction. As the successive reductions occur, they isolate and eventually cover the pixels in the more uniform region interiors. The last pixels to be enveloped are chosen as region seeds. The growing process that follows is essentially the reverse of this region shrinking.

The order in which pixels are considered for merging is a major concern. Truly parallel "best first" growth can be implemented on a sequential machine only by expensive schemes to repeatedly examine eligible pixels. Single scan methods have been proposed [Yakimovsky76, Somerville76], although a second scan is necessary to label the region map.

Deciding whether to merge a pixel with an adjacent region is equivalent to a one-sided hypothesis test. Some measure of membership must be computed and some threshold must be used. Often the pixel is compared with the region mean, using the region variance to set a threshold. Somerville and Mundy [Somerville76] use a planar approximation to the region, thus allowing for slope in the luminance function. Other researchers have compared the unassigned pixel only to the region pixels nearest it. For a survey of techniques see Zucker [Zucker76a].

A major problem with region growing is leakage, similar to chaining in cluster analysis. Two very dissimilar regions may be joined by an area of intermediate appearance: it is then possible for one region to grow across the neck and absorb pixels belonging to the other region. This can be remedied by recursive splitting or by a split-and-merge editing phase, but greatly complicates the segmentation process.

A.8. Merging

Another approach is region merging, beginning with uniform or single pixel regions. Those regions sharing a common border are eligible for merging. The border is eliminated if the combined region is sufficiently homogeneous. This differs from region growing in that both regions to be merged may be larger than one pixel.

The decision of whether to merge two regions can be based on the strength of the boundary between them. This leads to trouble when two distinct regions share a blurred or indistinct border. Merging can also be treated as a hypothesis test: the two regions are combined only if this gives an acceptable planar fit to the data.

The results of region merging may depend strongly on the order in which region pairs are tested for merging. Order independence may be achieved by considering all merges in parallel and allowing only the best merge to occur at any one time. This requires extra computation and "bookkeeping," leading many investigators to develop approximations to best-first merging.

Merging algorithms avoid recomputation if the uniformity measure for a combined region is a function of the statistics of its subregions. Maximum and minimum pixel values, for instance, can be computed from the subregion extrema: only the initial N^2 pixel examinations are needed. Unfortunately a large number of storage locations (N^2 per feature in theory, but less in practice) are required to hold the region statistics. Elaborate data structures may also be required to keep track of the numerous

irregularly-shaped regions.

Semantic merging integrates segmentation with interpretation. Yakimovsky and Feldman [Yakimovsky73a, Yakimovsky73b, Feldman74] suggest using real-world probabilities of region-type adjacencies. Such probabilities may be obtainable for limited domains such as X-ray analysis. A similar approach to region labeling has been proposed by Tenenbaum and Barrow [Barrow76, Tenenbaum76a, Tenenbaum76b].

A.9. Split-Merge

Most researchers using splitting or merging techniques alone have acknowledged the need for the complementary process as an editing step. At any stopping point in a segmentation there are usually some regions that should be split further and some that should be merged.

Merging techniques generally consider only two subregions at a time, and the final partitioning depends on the order of these comparisons. Splitting techniques are similarly limited by the order in which histogram peaks are chosen. The best possible partioning, by any particular criterion, might not be reachable by either technique. Integrated (or iterated) splitting and merging may also fall short of this ideal, but the combination is able to explore a larger space of possibilities.

Split-merge methods do not require accurate region seeds. Horowitz and Pavlidis [Horowitz74] start with arbitrary square neighborhoods. (This is particularly useful for computing Fourier texture measures over the seed regions [Pavlidis75].) Their algorithm breaks the nonuniform squares into uniform seeds, then combines neighboring fragments that are similar. The similarity measure may be based on intensity or on texture properties [Chen79]. No connected-components analysis is necessary if a segmentation tree is maintained.

Split-merge methods are able to use local information to determine each splitting, but the region boundaries tend to "cling" to the major rectilinear divisions. The splitting steps integrate well with quadtree representation of segmentation maps [Horowitz74, Klinger76, Hunter79, Samet79], but a merging step tends to destroy the quadtree structure. More elaborate linked tree structures have been developed [Burt80, Pietikäinen82] to solve this problem.

Although these methods have become strongly linked to quadtree representations, it is important to note that a split-merge approach is compatible with chain-code outlines, binary overlays, region maps, or other representations.

A.10. Spanning-Tree Methods

Several researchers have proposed tree structures to model the hierarchical structure of a scene. (Often neighbor relationships are stored, making the structure a graph rather than a tree.) The root node is the image itself; leaf nodes are the individual pixels or homogeneous regions. The scene may be segmented at any resolution by cutting branches of the tree [Kirsch71, Freuder76, Horowitz76, Horowitz78].

Burr and Chien [Burr76] apply minimal spanning tree methods to find strongly linked pixel groups separated from each other by weak links. The one-pass segmentation method of Yakimovsky [Yakimovsky76] builds a spatially constrained approximation

to the minimal spanning tree; it could be called a minimal spanning maze. A very similar segmentation system is developed in Narendra [Narendra77, Narendra80].

The spanning-tree methods all require that region interiors be smoother than border neighborhoods. They are thus unsuitable for locating textured regions unless the textures can be transformed to one or more feature planes with the property of region homogeneity. Macrotextures must be analyzed by identifying the primitive elements, then using structural methods to find texture regions.

A.11. Segmentation Editing

The preceding methods provide the best segmentation possible using local statistical analysis. The purpose of an editing phase is to improve the segmentation by using more global or application-dependent knowledge.

It is much easier to merge regions than to split them, since splitting requires that the best boundary be identified. Images are thus nearly always oversegmented to simplify the editing or interpretation phases that follow.

Syntactic editing analyzes the properties of regions and their spatial relationships. Pavlidis *et al.* [Tanimoto77, Horowitz78] use region adjacency graphs to identify noise regions. These are deleted and the pixels are reassigned to neighboring regions. Riseman and Arbib [Riseman77] use region adjacency graphs to identify composite textures. The regions are considered texture elements, and it is desired to find larger regions containing distinctive distributions of these primitives.

One of the main reasons for segmentation of textured images is to permit region-by-region classification, which should be more accurate than pixel-by-pixel methods. The classification can be done using multispectral discriminant analysis [Gupta74], cluster analysis on within-region textures [Lumia81], or model-based shape analysis [Brenner77, Pavlidis78, Jain80, Rutkowski81]. Primitive regions can then be merged if their signatures are classified identically.

Semantic merging integrates region growing with interpretation [Yakimovsky73a, Yakimovsky73b, Feldman74, Barrow76, Garvey76a, Garvey76b, Tenenbaum76a, Tenenbaum76b, Tenenbaum80, Fischler82]. Probabilities of region-type adjacencies may be even more applicable at the final editing and classification stage [Lumia81].

Another form of editing uses initial region knowledge to guide a more sophisticated segmenter. It may be possible, by examining the initial edge and interior points, to infer a classifying rule or grammar [Keng77a, Keng77b]. This *bootstrap* information can then be used to resegment the scene or to segment other similar scenes. Bootstrapping is particularly effective if ground-truth segmentations are used to infer the rules.

There is no reason why editing must be limited to a single pass. Iterative parallel algorithms have been suggested [Rosenfeld76b, Riseman77] in which each pixel's label or region membership is repeatedly updated as a function of its neighbors' labels. These competitive-cooperative processes have also been used for edge thinning and edge linking [Zucker76b, Zucker77]. The methods are very flexible and powerful, but little is known about constructing the label assignment functions.

After a scene has been segmented into regions, it is still necessary to determine which of these regions belong to composite objects. Even a simple object such as an

untextured block may have several distinct regions because of lighting effects. On the other hand, a single uniform region may be segmentable into a stack of blocks or a clump of particles on the basis of its outline [Arcelli71]. (The Price segmenter performs some shape analysis and region editing during connected-component extraction. This is a rather expensive step, and PHOENIX has left it for an external editing program.)

There have been many attempts to combine segmentation with semantic interpretation in natural scenes; see, for instance, [Preparata72, Tenenbaum73, Feldman74, Barrow76, Garvey76a, Garvey76b, Price76b, Sakai76, Tenenbaum76a, Tenenbaum76b, Levine77, Faugeras80, Tenenbaum80, Price81, Fischler82]. Such recognition requires domain-specific knowledge beyond the scope of this study.

Appendix B

Connected Component Extraction

The following information on the connected component extraction algorithm was provided by Duane Williams of Carnegie-Mellon University as part of the PHOENIX code. (For the algorithm used in the Ohlander/Price segmenter, see [Ohlander78]. Another algorithm is given in Kelly [Kelly70, pp. 54-55].)

This algorithm is the connected region extraction algorithm, *reganal*, developed for the KIWI segmentation program at Carnegie-Mellon University. It is based upon the method of Agrawala and Kulkarni [Agrawala77]. This implementation (and that of KIWI) differs, however, in several points from their algorithm.

This algorithm takes a binary image, and produces a list of descriptions of the component regions (patches) and their pixels (strips). The patches are represented by *patch* records, and include shape features and an indication of which patch contains this one (*i.e.*, surrounds it). The strips are described by *strip* records, which include a row, the columns on which the strip begins and ends, and a link to the next strip. The input image is actually a map of the interval numbers resulting from thresholding; this procedure is executed once for each interval, and considers pixels in that interval (given by the parameter *val*) to be '1,' all others to be '0.' A border of 0's is assumed to surround the image.

The algorithm proceeds by forming, for each row, a description of the strips of that row. This description includes, for each consecutive run of 1's or 0's, the column at which the run starts. The run ends one column before the next run starts. The runs for each row are compared with the runs of the previous row, by examining the locations of the endpoints, to determine how to propagate partial region labels from the previous row to the current row. The examination is performed by the *assign* procedure. This procedure can perform five actions: create a new region of 1's (newbody), create a new region of 0's (newhole), propagate a label from an existing region (extend), end a region of 1's (endbody), end a region of 0's (endhole).

The actions in *assign* take place within a big loop that scans one *segment* (run of 1's followed by a run of 0's) in the previous row, dealing with all segments in the current row that are encountered. At their leftmost endpoint, the runs in the current row are labeled. This big loop may encounter eight situations: four while scanning the 0's before the ones, and four while scanning the 1's before the next 0. Here, pictorially, are the possible situations; the letters A, B, *etc.*, mark the start of the next run; + indicates a 1 and - indicates a 0:

```
Case 1:
          Prev. Row:    ...+++A----------B++++...
          This row:     ...+++++++W------...
          Description:  a run of 1's extends from before A, into the
                                hole AB.
          Actions:      extend (AB) to (W...)
          Reasons:      the hole W... Touches the hole AB;
                                the run ...W has already been labeled.
```

```
Case ii:
                prev. Row:      ...+++A------------B++++...
                This row:       ...++++++++++++++++++++...
                Description:    a run of 1's extends from before A until
                                        somewhere past B.
                Actions:        endhole (AB)
                reasons:        the hole AB cannot continue below; the
                                        run on this row has already been labeled.

Case iii:
                prev. Row:      ...+++A------------B++++...
                This row                ....---V++W---...
                Description:    a run of 1's begins and ends within the
                                        run AB.
                Actions:        newbody (VW); extend (AB) to (W...)
                Reasons:        the body VW is created; the hole W...
                                        Touches the hole AB.

Case iv:
                prev. Row:      ...+++A------------B++++...
                This row:                       ...---V+++++++...
                Description:    a run of 1's starts between A and B, and
                                        continues past B
                Actions:        extend (B...) to (V...)
                Reasons:        the body V... Touches the body B...
                                        The hole ...V has already been labeled.

Case v:
                prev. Row:      ...---B++++++++++++C----...
                This row:       ...------V+++....
                Description:    a hole starts before B, and ends before C.
                Actions:        extend (BC) to (V...)
                Reasons:        the body V... Touches the body BC; the hole
                                        ...V has already been labeled.

Case vi:
                prev. Row:      ...---B++++++++++++C----...
                This row:       ...-------------------------...
                Description:    a hole starts before B and ends after C.
                Actions:        endbody (BC)
                reasons:        the body BC does not continue below; the hole
                                        on this row has already been labeled.

Case vii:
                prev. Row:      ...---B++++++++++++C----...
                This row:                       ..++U---V++...
                Description:    a hole starts and ends within the run BC.
                Actions:        newhole (UV); extend (BC) to (V...)
                Reasons:        the hole UV is new; the body V... touches BC.

Case viii:
                prev. Row:      ...---B++++++++++++C----...
                This row:                       ...++U----------...
                Description:    a hole starts between B and C, and ends
                                        after C.
                Actions:        extend (C...) to (U...)
                Reasons:        the body ...U has already been labeled; the
                                        hole U... Touches the hole C...
```

There are two cases not covered here. the case in which a hole starts before A and ends after B, and the case in which a run of 1's starts before B and ends after C. These cases need not be examined, since they involve no new bodies or holes, no ending bodies or holes, and no propagation of labels.

There are also two cases not completely examined, cases ii and vi, in which a body or hole ends. In case ii, we must note the fact that body B... is touching the run on the current row, which is touching body two possibilties. If ...A and B... are the same partial region then the hole AB is completely contained within that partial region, if ...A and B... are different, they are to be merged together. Both cases are discussed below in more detail. Similarly, in case vi, holes ...B and C... touch each other. In the program, these cases are handled by the *endbody* and *endhole* procedures.

It must also be kept in mind that a single partial region may, by cases iii and vii, be split up into any number of runs on a single row; some of these may be ended, some merged, and some extended on any given row of the image. So, we must keep track of exactly what has happened to a partial region throughout the scan of the entire row; then, we can do drastic things (like declaring a partial region to be really at its end) at the end of the row.

There is always an issue, in connectivity algorithms, of the exact definition of connectivity. Two definitions are the most common: 4-connectivity and 8-connectivity.

The definitions are these:

```
4-connected      8-connected
    x              xxx
   x+x             x+x
    x              xxx
```

(the pixel + is connected to all pixels x.)

For reasons pointed out by Rosenfeld [Rosenfeld76a], it is frequently desirable to have objects (*i.e.*, 1's) be 4-connected and holes (*i.e.*, 0's) 8-connected, or vice versa. In fact, this algorithm depends on this distinction. For the segmentation program, it is necessary to have objects be 4-connected in order to avoid some infinite-loop situations, for example, if the input is alternating 1 and 0 pixels, like a checkerboard. So, holes are 8-connected and objects are 4-connected. This may be reversed (objects 8-connected and holes 4-connected) by converting all the '<' signs in *assign* (where column numbers are being compared) to '<=,' and all the '<=' signs (again, only for comparisons of column numbers) to '<'.

A single row is represented by the *line* data structure. This contains the number of segments, *Lsegs*; the segments *Lseg* themselves; and two counters: *Lcurseg* and *Lcol*. These counters are used in *assign* for indicating the *current* segment (Lcurseg) and the column on which the next segment begins (Lcol). Each segment record indicates the column of the first 1, the column of the first 0; and the partial-region labels assigned to the 1's and the 0's.

There is assumed to be a region of 0's surrounding the image; this is called *outside*, and is represented by partial region *preg_outside*. This is accomplished by the following steps:

- The *firstrow* procedure pretends there is a row of 0's from before the first column until past the last one.
- The *runcode* procedure pretends there are 0's from the last 1 until past the end of the image.
- The *extend* procedure pretends there is another segment to the left of the first segment of the row, which has already been labeled as *outside*.
- The merging procedure (*etc.*) always merges other partial regions into *outside*; never *outside* into another partial region.
- The region description for the *outside* region is not meaningful, and may contain garbage.
- The *lastrow* procedure pretends there is a r)'s from before the first column until past the last one.

Peculiarities of PR_desc:

The fields of the region description of a partial region (*PR_desc*) are used in a special way: *R_rstart* is indeed the first row of the partial region. *R_rows*, however, is the last row number rather than the number of rows. Similarly, *R_cstart* is the starting column, but *cols* is the ending column. *R_area*, *R_holes*, and *R_harea* are normal. The centroid, however, is accumulated in *R_rcent* and *R_ccent* as the sum of the row (and column) number for each pixel of the partial region. Then, when the patch record is written, *R_rcent* is divided by *R_area* (also, *R_ccent* is divided by *R_area*) to compute the actual coordinates of the centroid.

The most important piece of information in the patch record is the link to the containing patch, *P_outer*. However, as stated in the paragraph above, the patch record is created when its partial region(s) comes to an end; this is before the patch record for the containing patch has been created! So, it is necessary to remember, when a patch is created, which partial region contains it. Then, when a patch is made from this partial region, the link in the contained patch can be updated. This remembering is accomplished via the *PR_inner* and *P_next* links: each partial region points (via *P_inner*) to a patch it contains, which points (via *P_next*) to the next one, and so on. When the partial region is converted to a patch, this list is scanned, and the new patch number is placed into the *P_outer field.*

There is one problem with the above structure: when partial region A is merged into partial region B, both A and B have these lists of contained patches. The lists could be combined by traversing one list and updating the link of the last patch, *etc.* However, the lists may become quite long, and it is not attractive to have to scan through them (potentially many times, as partial regions are merged). So, instead, each partial region has a list of other partial regions that have been merged into it (*PR_piece*), with the last partial region on the list containing *PREG_NIL* as its *PR_piece* field. When a partial region is converted to a patch, this list is traversed and all the patches contained by all these partial regions are updated. The partial regions may then be freed so they may be used again.

There is, however, a further problem. Since all these merged partial regions are kept around, there may be references to them (*i.e.*, segments that are labeled with these merged partial regions, other partial regions indicating that these partial regions surround them, *etc.*). So, whenever such a reference is made, it is necessary to find which partial region is really indicated (thus, if A is merged into B and we refer to A, we really want to talk about B). The *PR_whole* field is a link to the partial region used after merging, and the *root* function traces down these links to find the intended partial region. Note that *PR_piece* is not the exact inverse of *preg_whole*. The *PR_whole* fields form a list from the active partial region through all those partial regions merged with it. If, however, A is merged with B and B is merged with C, then the *PR_whole* field of A points to B and *PR_whole* of B points to C. If, then, D is merged into C, PR_whole of D also points to C. In this example, the *PR_piece* fields form a real linked list:

```
C -> D -> B -> A
```

while the *PR_whole* fields form a tree:

```
A
!
B D
!/
C
```

(with links pointing down, in this picture).

The run codes normally indicate a row, the columns at which the run starts and ends,

and a link to the next run for the same region. When partial regions are merged, they each indicate a linked list of runs; somehow, these must be merged as well. This is accomplished by a special run whose rcw number is the special value *S_MERGE*. This run has two fields: pointers to the two linked lists to be merged. During the actual traversal of the runs, both lists must be examined when a merge run is encountered.

The column numbers used in this procedure are sometimes tricky. Normally, for each run of 1's and 0's (*i.e.*, in the segment record), the column of the start of each run is stored. This means that the last column of a run of 1's is actually the start of the next run of 0's, minus one. In the partial region records, *cols* is this value; actually, one plus the rightmost column of the partial region. When patch records are created, the proper conversion is performed. Also, when run records are stored, the column of the end of the run is really the last column of the run; *i.e.*, 1 has already been subtracted.

References

[Abdou79] I.E. Abdou and W.K. Pratt, "Quantitative Design and Evaluation of Enhancement/Thresholding Edge Detectors," *Proc. IEEE*, Vol. 67, No. 5, pp. 753-763, May 1979.

[Aggarwal77] R.K. Aggarwal and J.W. Bacus, "A Multi-Spectral Approach for Scene Analysis of Cervical Cytology Smears," *J. Histochem. Cytochem.*, Vol. 25, pp. 668-680, 1977.

[Aggarwal78] R.K. Aggarwal, "Adaptive Image Segmentation using Prototype Similarity," *Proc. IEEE Conf. on Pattern Rec. and Image Processing*, Chicago, pp. 354-359, 1978.

[Agrawala77] A.K. Agrawala and A.V. Kulkarni, "A Sequential Approach to the Extraction of Shape Features," *Computer Graphics and Image Processing*, Vol. 6, pp. 538-557, 1977.

[Ahuja77] N. Ahuja, L.S. Davis, R.M. Haralick, and D.P. Panda, *Image Segmentation Based on Local Gray Level Patterns*, Report TR-551, U. of Maryland, College Park, June 1977.

[Ali79] M. Ali, W.N. Martin, and J.K. Aggarwal, "Color-Based Computer Analysis of Aerial Photographs," *Computer Graphics and Image Processing*, Vol. 9, pp. 282-293, 1979.

[Arcelli71] C. Arcelli and S. Levialdi, "Picture Processing and Overlapping Blobs," *IEEE Trans. on Computers*, pp. 1111-1115, Sep. 1971.

[Barrett81] W.A. Barrett, "An Iterative Algorithm for Multiple Threshold Detection," *Proc. IEEE Conf. on Pattern Rec. and Image Processing*, Dallas, pp. 273-278, Aug. 1981.

[Ballard82] D.H. Ballard and C.M. Brown, *Computer Vision*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1982.

[Barrow75] H.G. Barrow and J.M. Tenenbaum, *Representation and Use of Knowledge in Vision*, TN 108, Artificial Intelligence Center, SRI International, July 1975.

[Barrow76] H.G. Barrow and J.M. Tenenbaum, *MSYS: A System for Reasoning about Scenes*, TN 121, Artificial Intelligence Center, SRI International, Apr. 1976.

[Barrow81] H.G. Barrow and J.M. Tenenbaum, "Computational Vision," *Proc. IEEE*, Vol. 69, No. 5, pp. 572-595, May 1981.

[Bhanu82] B. Bhanu and O.D. Faugeras, "Segmentation of Images Having Unimodal Distributions," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-4, No. 4, pp. 408-419, July 1972.

[Brady82] M. Brady, "Computational Approaches to Image Understanding," *Computing Surveys*, Vol. 14, No. 1, pp. 3-71, Mar. 1982.

[Brenner77] J.F. Brenner *et al.*, "Scene Segmentation Techniques for the Analysis of Routine Bone Marrow Smears from Acute Lymphoblastic Leukemia Patients," *J. Histochem. Cytochem.*, Vol. 25, pp. 601-613, 1977.

[Brice70] C.R. Brice and C.L. Fennema, "Scene Analysis using Regions," *Artificial Intelligence*, Vol. 1, pp. 205-226, Fall 1970.

[Bullock76] B.L. Bullock, "Finding Structure in Outdoor Scenes," in C.H. Chen (ed.), *Pattern Recognition and Artificial Intelligence*, Academic Press, New York, 1976.

[Burr76] D.J. Burr and R.T. Chien, "The Minimal Spanning Tree in Visual Data Segmentation," *Proc. 3rd Int. Jnt. Conf. on Pattern Recognition*, Coronado, CA, pp. 519-523, Nov. 1976.

[Burt80] P. Burt, T.H. Hong, and A. Rosenfeld, *Segmentation and Estimation of Image Region Properties through Cooperative Hierarchical Computation*, TR-927, Computer Vision Laboratory, Computer Science Center, Univ. of Maryland, College Park, Aug. 1980.

[Cahn77] R.L. Cahn, R.S. Poulsen, and G. Toussaint, "Segmentation of Cervical Cell Images," *J. Histochem. Cytochem.*, Vol. 25, pp. 681-688, 1977.

[Carlton77] S.G. Carlton and O.R. Mitchell, "Image Segmentation using Texture and Gray Level," *Proc. Image Understanding Workshop*, pp. 71-77, Apr. 1977.

[Chen79] P.C. Chen and T. Pavlidis, "Segmentation by Texture Using a Co-Occurrence Matrix and a Split-and-Merge Algorithm," *Computer Graphics and Image Processing*, Vol. 10, pp. 172-182, 1979.

[Chow70] C.K. Chow and T. Kaneko, *Boundary Detection of Radiographic Images by a Threshold Method*, IBM Research Report RC-3203, 1970. Also in *Proc. IFIP 71*, TA-7, p. 130, 1971, and in S. Watanabe (ed:), *Frontiers of Pattern Recognition*, Academic Press, NY, pp. 61-82, 1972.

[Clark81] S.J. Clark, *Image File Naming Conventions*, IUS Document CMU004, Computer Science Dept., Carnegie-Mellon Univ., Mar. 1981.

[Coleman79] G.B. Coleman and H.C. Andrews, "Image Segmentation by Clustering," *Proc. IEEE*, Vol. 67, No. 5, pp. 773-785, May 1979.

[Danker81] A.J. Danker and A. Rosenfeld, "Blob Detection by Relaxation," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-3, No. 1, pp. 79-92, Jan. 1981.

[Davis75] L.S. Davis, "A Survey of Edge Detection Techniques," *Computer Graphics and Image Processing*, Vol. 4, No. 1, pp. 248-270, Sep. 1975.

[Davis80] L.S. Davis and A. Mitiche, "Edge Detection in Textures," *Computer Graphics and Image Processing*, Vol. 12, No. 1, pp. 25-39, Jan. 1980.

[Davis82] L.S. Davis and A. Mitiche, "MITES (mit-es): A Model-Driven, Iterative Texture Segmentation Algorithm," *Computer Graphics and Image Processing*, Vol. 19, pp. 95-110, 1982.

[Deal79] B. Deal, C.M. Lo, R. Taylor, V. Norwood, H. Henning, T. Daggett, T. Noda, J. Powers, G. Guzman, H. Greenberger, G. Towner, and G. Parker, *Automatic Target Cuer First Quarter Report*, Northrop Corp., Electro-Mechanical Division, Anaheim, CA, Report NORT-79Y100, Oct. 1979.

[Deguchi78] K. Deguchi and I. Morishita, "Texture Characterization and Texture-Based Image Partitioning Using Two-Dimensional Linear Estimation Techniques," *IEEE Trans. on Computers*, Vol. C-27, No. 8, pp. 739-745, Aug. 1978.

[Dyer81] C.R. Dyer, "A VLSI Pyramid Machine for Hierarchical Parallel Image Processing," *Proc. IEEE Conf. on Pattern Rec. and Image Processing*, Dallas, pp. 381-386, Aug. 1981.

[Ehrich76] R. Ehrich and J.P. Foith, "Representation of Random Waveforms by Relational Trees," *IEEE Trans. on Computers*, Vol. C-25, pp. 725-736, July 1976.

[Eklundh80] J.O. Eklundh, H. Yamamoto, and A. Rosenfeld, "A Relaxation Method for Multispectral Pixel Classification," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-2, No. 1, pp. 72-75, Jan. 1980.

[Faugeras80] O.D. Faugeras and K.E. Price, "Semantic Description of Aerial Images Using Stochastic Labeling," *Proc. ARPA Image Understanding Workshop*, pp. 89-94, Apr. 1980. Also in *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-3, No. 6, pp. 633-642, Nov. 1981.

[Feldman74] J.A. Feldman and Y. Yakimovsky, "Decision Theory and Artificial Intelligence: I. A Semantics-Based Region Analyzer," *Artificial Intelligence*, Vol. 5, pp. 349-372, Winter 1974.

[Fischler79] M.A. Fischler, G.J. Agin, H.G. Barrow, R.C. Bolles, L. Quam, J.M. Tenenbaum, and H.C. Wolf, *Interactive Aids for Cartography and Photo Interpretation*, Final Technical Report, Artificial Intelligence Center, SRI International, 1979.

[Fischler82] M.A. Fischler, S.T. Barnard, R.C. Bolles, M. Lowry, L. Quam, G. Smith, and A. Witkin, *Modeling and Using Physical Constraints in Scene Analysis*, TN 267, Artificial Intelligence Center, SRI International, Sep. 1982.

[Fischler83] M.A. Fischler and H.C. Wolf, *A General Approach to Machine Perception of Linear Structure in* Imaged Data," TN 276, Artificial Intelligence Center, SRI International, Feb. 1983.

[Fram75] J.R. Fram and E.S. Deutsch, "On the Quantitative Evaluation of Edge Detection Schemes and their Comparison with Human Performance," *IEEE Trans. on Computers*, Vol. C-24, No. 6, pp. 616-628, June 1975.

[Freuder76] E. Freuder, "Affinity: A Relative Approach to Region Growing," *Computer Graphics and Image Processing*, Vol. 5, pp. 254-264, 1976.

[Fu81] K.S. Fu and J.K. Mui, "A Survey on Image Segmentation," *Pattern Recognition*, Vol. 13, pp. 3-16, 1981.

[Garvey76a] T.D. Garvey, *Perceptual Strategies for Purposive Vision*, TN 117, Artificial Intelligence Center, SRI International, Sep. 1976.

[Garvey76b] T.D. Garvey and J.M. Tenenbaum, *Application of Interactive Scene Analysis Techniques to Cartography*, TN 127, Artificial Intelligence Center, SRI International, Sep. 1976.

[Goldberg76] M. Goldberg and S. Shlien, "A Cluster Scheme for Multispectral Images," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-8, pp. 86-92, 1978.

[Grimson80] W.E.L. Grimson, "Aspects of a Computational Theory of Human Stereo Vision," *Proc. Image Understanding Workshop*, College Park, MD, pp. 128-149, Apr. 1980.

[Gupta74] J.N. Gupta and P.A. Wintz, "Computer Processing Algorithm for Locating Boundaries in Digital Pictures," *Proc. 2nd Int. Jnt. Conf. on Pattern Recognition*, Copenhagen, pp. 155-156, Aug. 1974.

[Gurari82] E.M. Gurari and H. Wechsler, "On the Difficulties Involved in the Segmentation of Pictures," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-4, No. 3, pp. 304-306, May 1982.

[Hanson74] A.R. Hanson and E.M. Riseman, *Preprocessing Cones: A Computational Structure for Scene Analysis*, Computer and Information Science Report 74C-7, Univ. of Mass. at Amherst, Sep. 1974.

[Hanson75a] A.R. Hanson and E.M. Riseman, *The Design of a Semantically Directed Vision Processor*, Computer and Information Science Report 75C-1, Univ. of Mass. at Amherst, Feb. 1975.

[Hanson75b] A.R. Hanson, E.M. Riseman, and P. Nagin, *Region Growing in Textured Outdoor Scenes*, Computer and Information Science Report 75C-3, Univ. of Mass. at Amherst, Feb. 1975.

[Hanson78] A.R. Hanson and E.M. Riseman, "Segmentation of Natural Scenes," in A.R. Hanson and E.M. Riseman (eds.), *Computer Vision Systems*, New York: Academic Press, 1978. See also color plate 5-4 in [Ballard82].

[Haralick73] R.M. Haralick, K. Shanmugam, and I. Dinstein, "Textural Features for Image Classification," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-3, pp. 610-621, Nov. 1973.

[Haralick75a] R.M. Haralick and I. Dinstein, "A Spatial Clustering Procedure for Multi-Image Data," *IEEE Trans. on Circuits and Systems*, Vol. CAS-22, pp. 440-450, 1975.

[Haralick75b] R.M. Haralick, "A Resolution Preserving Textural Transform for Images," *Proc. Computer Graphics, Pattern Recognition, and Data Structure*, Los Angeles, pp. 51-61, May 1975.

[Haralick76] R.M. Haralick, "Automatic Remote Sensor Image Processing," in A. Rosenfeld (ed.), *Digital Picture Analysis*, Berlin, Germany: Springer, pp. 5-63, 1976.

[Haralick80] R.M. Haralick, "Edge and Region Analysis for Digital Image Data," *Computer Graphics and Image Processing*, Vol. 12, pp. 60-73, 1980.

[Haralick81] R.M. Haralick, "The Digital Edge," *IEEE Conf. on Pattern Rec. and Image Processing*, Dallas, pp. 285-291, Aug. 1981.

[Harlow73] C.A. Harlow and S.A. Eisenbeis, "The Analysis of Radiographic Images," *IEEE Trans. on Computers*, Vol. C-22, pp. 678-688, 1973.

[Horn77] B.K.P. Horn, "Understanding Image Intensity," *Artificial Intelligence*, Vol. 8, pp. 201-231, 1977.

[Horowitz74] S.L. Horowitz and T. Pavlidis, "Picture Segmentation by a Directed Split-and-Merge Procedure," *Proc. 2nd Int. Jnt. Conf. on Pattern Recognition*, Copenhagen, pp. 424-433, Aug. 1974.

[Horowitz76] S.L. Horowitz and T. Pavlidis, "Picture Segmentation by a Tree Traversal Algorithm," *J. of the ACM*, Vol. 23, No. 2, pp. 368-388, Apr. 1976.

[Horowitz78] S.L. Horowitz, "A Graph-Theoretic Approach to Picture Processing," *Computer Graphics and Image Processing*, Vol. 7, No. 2, pp. 282-291, Apr. 1978.

[Hueckel71] M.H. Hueckel, "An Operator which Locates Edges in Digitized Pictures," *J. of the ACM*, Vol. 18, No. 1, pp. 113-125, Jan. 1971.

[Hueckel73] M.H. Hueckel, "A Local Visual Operator which Recognizes Edges and Lines," *J. of the ACM*, Vol. 20, No. 4, 634-647, Oct. 1973.

[Hummel78] R.A. Hummel and A. Rosenfeld, "Relaxation Processes for Scene Labeling," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-8, No. 10, pp. 765-768, Oct. 1978.

[Hunter79] G.M. Hunter and K. Steiglitz, "Operations on Images using Quad Trees," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-1, No. 2, pp. 145-153, Apr. 1979.

[Jain80] A.K. Jain, S.P. Smith, and E. Backer, "Segmentation of Muscle Cell Pictures: A Preliminary Study," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-2, No. 3, pp. 232-242, May 1980.

[Jarvis75] R.A. Jarvis, "Image Segmentation by Interactively Combining Line, Region, and Semantic Structure," *Proc. Conf. on Computer Graphics, Pattern Recognition, and Data Structures*, Los Angeles, pp. 279-288, May 1975.

[Kanade80] T. Kanade, "Region Segmentation: Signal vs Semantics," *Computer Graphics and Image Processing*, Vol. 13, pp. 279-297, 1980. Reprinted in Y.H. Pao and G.W. Ernst (eds.), *Context-Directed Pattern Recognition and Machine Intelligence Techniques for Information Processing*, pp. 518-533, 1982.

[Kasvand74] T. Kasvand, "Segmentation of Single Gray Level Pictures of General 3D Scenes," *Proc. 2nd Int. Jnt. Conf. on Pattern Recognition*, Copenhagen, pp. 372-373, Aug. 1974.

[Kelly70] M.D. Kelly, *Visual Identification of People by Computer*, Report AIM-130, Computer Science Thesis, Stanford Univ., July 1970.

[Kelly71] M.D. Kelly, "Edge Detection in Pictures by Computer using Planning," in B. Meltzer and D. Michie (eds.), *Machine Intelligence*, Vol. 6, Edinburgh Univ. Press, pp. 397-409, 1971.

[Kender76] J.R. Kender, *Saturation, Hue, and Normalized Color: Calculation, Digitization Effects, and Use*, Dept. of Computer Science, Carnegie-Mellon Univ., Nov. 1976.

[Kender77] J.R. Kender, "Instabilities in Color Transformations," *IEEE Conf. on Pattern Rec. and Image Processing*, Troy, NY, pp. 266-274, June 1977.

[Keng77a] J. Keng, "Image Segmentation and Object Detection by a Syntactic Method," *Proc. Image Understanding Workshop*, Minneapolis, pp. 38-43, Apr. 1977.

[Keng77b] J. Keng, *Syntactic Algorithms for Image Segmentation and a Special Computer Architecture for Image Processing*, Ph.D. Thesis, Purdue Univ., West Lafayette, Indiana, Dec. 1977.

[Kettig76] R.K. Kettig and D.A. Landgrebe, "Classification of Multispectral Image Data by Extraction and Classification of Homogeneous Objects," *IEEE Trans. on Geosci. Electron.*, Vol. GE-14, pp. 19-25, Jan. 1976.

[Kirby79] R. Kirby, "A Product Rule Relaxation Method," *Computer Graphics and Image Processing*, Vol. 10, pp. 235-245, 1979.

[Kirsch71] R.A. Kirsch, "Computer Determination of the Constituent Structure of Biological Images," *Computers and Biomedical Research*, Vol. 4, No. 3, pp. 315-328, June 1971.

[Klein77] G.M. Klein and S.A. Doudani, "Locating Man-Made Objects in Low-Resolution Outdoor Scenes," *Applic. of Digital Image Processing*, SPIE Vol. 119, pp. 278-283, 1977.

[Klinger76] A. Klinger and C.R. Dyer, "Experiments on Picture Representation using Regular Decomposition," *Computer Graphics and Image Processing*, Vol. 5, pp. 68-105, 1976.

[Kohler81] R. Kohler, "A Segmentation System Based on Thresholding," *Computer Graphics and Image Processing*, Vol. 15, pp. 319-338, 1981.

[Laws80] K.I. Laws, *Textured Image Segmentation*, USC Image Processing Inst., Los Angeles, CA, Report USCIPI 940, Jan. 1980.

[Laws83] K.I. Laws, *The GHOUGH Generalized Hough Transform Package: Description* and Evaluation," Technical Note, Artificial Intelligence Center, SRI International, June 1983.

[Lee82] C.H. Lee, "Iterative Region Segmentation," *IEEE Conf. on Pattern Rec. and Image Processing*, Las Vegas, pp. 557-559, June 1982.

[Lemkin79] P. Lemkin, "An Approach to Region Splitting," *Computer Graphics and Image Processing*, Vol. 10, No. 3, pp. 281-288, July 1979.

[Levine76] M.D. Levine and J. Leemet, "A Method for Non-Purposive Picture Segmentation," *Proc. 3rd Int. Jnt. Conf. on Pattern Recognition*, Coronado, CA, pp. 494-498, Nov. 1976.

[Levine77] M.D. Levine, *A Knowledge-Based Computer Vision System*, Report R-77-3, Dept. of Electrical Engineering, McGill Univ., Oct. 1977. See also the same title in *Adv. papers for the Workshop on Computer Vision Systems, Vol. III*, Univ. of Mass., pp. 1-20, June 1977.

[MacVicar-Whelan81] P.J. MacVicar-Whelan and T.O. Binford, "Line Finding with Subpixel Precision," *Proc. Image Understanding Workshop*, Washington, D.C., pp. 26-31, Apr. 1981.

[Martelli76] A Martelli, "An Application of Heuristic Search Methods to Edge and Contour Detection," *Comm. of the ACM*, Vol. 19, No. 2, pp. 73-83, Feb. 1976.

[McCormick75] B.H. McCormick and S.N. Jayaramamurthy, "A Decision Theory Method for the Analysis of Texture," *Int. J. of Computer and Info. Sciences*, Vol. 4, No. 1, pp. 1-38, Mar. 1975.

[McKeown81] D. McKeown and J. Denlinger, *Grinnell Display Software Support*, IUS Document CMU002, Second Ed., Computer Science Dept., Carnegie-Mellon Univ., June 1981.

[Milgram77] D.L. Milgram, "Region Extraction using Convergent Evidence," *Proc. Image Understanding Workshop*, Minneapolis, pp. 58-64, Apr. 1977. Also Report TR-674, U. of Maryland Computer Science Center, June 1978, and in *Computer Graphics and Image Processing*, Vol. 11, pp. 1-13, 1979.

[Milgram78] D.L. Milgram, "Edge Point Linking using Convergent Evidence," *Proc. Image Understanding Workshop*, pp. 85-91, Nov. 1978.

[Milgram79] D.L. Milgram and D.J. Kahl, "Recursive Region Extraction," *Computer Graphics and Image Processing*, Vol. 9, pp. 82-88, 1979.

[Milgram80] D.L. Milgram and M. Herman, "Clustering Edge Values for Threshold Selection," *Computer Graphics and Image Processing*, Vol. 10, pp. 272-280, 1980.

[Mitchell78] O.R. Mitchell and S.G. Carlton, "Image Segmentation using a Local Extrema Texture Measure," *Pattern Recognition*, Vol. 10, pp. 205-210, 1978.

[Mitchell79] O.R. Mitchell, S.P. Lutton, and S.P. Su, "Texture Image Segmentation using Local Extrema," *Proc. IEEE Conf. on Pattern Rec. and Image Processing*, Chicago, pp. 508-511, Aug. 1979.

[Mitiche80] A. Mitiche and L. Davis, "Theoretical Analysis of Edge Detection in Textures," *Proc. 5th Int. Jnt. Conf. on Pattern Recognition*, Miami Beach, FL, pp. 540-547, Dec. 1980.

[Montanari71] U. Montanari, "On the Optimal Detection of Curves in Noisy Pictures," *Comm. of the ACM*, Vol. 14, No. 5, pp. 335-345, May 1971.

[Muerle68] J.L. Muerle and D.C. Allen, "Experimental Evaluation of Techniques for Automatic Segmentation of Objects in a Complex Scene," in G.C. Cheng *et al.* (eds.), *Pictorial Pattern Recognition*, Thompson, Washington, DC, pp. 3-13, 1968.

[Mui76] J.K. Mui, J.W. Bacus, and K.S. Fu, " Scene Segmentation Technique for Microscopic Cell Images," *Proc. Symp. Computer Aided Diagnosis of Medical Images*, San Diego, CA, IEEE CH1170-0C, pp. 99-106, 1976.

[Nagin77] P.A. Nagin, A.R. Hanson, and E.M. Riseman, *Region Extraction and Description through Planning*, Computer and Information Science Report 77-8, Univ. of Mass. at Amherst, May 1977.

[Nagin78] P.A. Nagin, *Segmentation using Spatial Context and Feature Space Cluster Labels*, Computer and Information Science Report 78-8, Univ. of Mass. at Amherst, May 1978.

[Nagin79] P. Nagin, R. Kohler, A. Hanson, and E. Riseman, "Segmentation, Evaluation, and Natural Scenes," *Proc. IEEE Conf. on Pattern Rec. and Image Processing*, Chicago, pp. 515-522, Aug. 1979.

[Nahi77] N.E. Nahi and M.H. Jahanshahi, "Image Boundary Estimation," *IEEE Trans. on Computers*, Vol. C-26, No. 8 pp. 772-781, Aug. 1977. Reprinted in H.C. Andrews (ed.), *Tutorial and selected papers in Digital Image Processing*, IEEE Computer Society, pp. 546-555, 1978.

[Nahi78] N.E. Nahi and S. Lopez-Mora, "Estimation-Detection of Object Boundaries in Noisy Images," *IEEE Trans. on Automatic Control*, Vol. AC-23, No. 5, pp. 834-846, Oct. 1978.

[Narendra77] P.M. Narendra and M. Goldberg, "A Graph-Theoretic Approach to Image Segmentation," *Proc.*IEEE Troy, NY, pp. 248-256, June 1977.

[Narendra80] P.M. Narendra and M. Goldberg, "Image Segmentation with Directed Trees," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-2, No. 2, pp. 185-191, Mar. 1980.

[Nevatia76] R. Nevatia, "Locating Object Boundaries in Textured Environments," *IEEE Trans. on Computers*, Vol. C-25, No. 11, pp. 1170-1175, Nov. 1976.

[Nevatia77a] R. Nevatia and K. Price, "A Comparison of some Segmentation Techniques", *Proc. Image Understanding Workshop*, pp. 55-57, Apr. 1977.

[Nevatia77b] R. Nevatia, "A Color Edge Detector and Its Use in Scene Segmentation," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-7, No. 11, pp. 820-826, Nov. 1977.

[Nevatia82] R. Nevatia, *Machine Perception*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[Ohlander75] R. Ohlander, *Analysis of Natural Scenes*, PhD. Thesis, Dept. of Computer Science, Carnegie-Mellon U., Pittsburg, Apr. 1975.

[Ohlander78] R. Ohlander, K. Price, and D.R. Reddy, "Picture Segmentation using a Recursive Region Splitting Method," *Computer Graphics and Image Processing*, Vol. 8, No. 3, pp. 313-333, Dec. 1978.

[Ohta80a] Y. Ohta, T. Kanade, and T. Sakai, "Color Information for Region Segmentation," *Computer Graphics and Image Processing*, Vol. 13, pp. 222-241, 1980.

[Ohta80b] Y. Ohta, *A Region-Oriented Image-Analysis System by Computer*, Ph.D. Thesis, Dept. of Info. Sciences, Kyoto Univ., Japan, Mar. 1980.

[O'Rourke81] J. O'Rourke, "Dynamically Quantized Spaces for Focusing the Hough Transform," *Proc. 7th Int. Jnt. Conf. on Artificial Intelligence*, Vol. 2, pp. 737-739, 1981.

[Panda78] D.P. Panda and A. Rosenfeld, "Image Segmentation by Pixel Classification in (Grey Level, Edge Value) Space," *IEEE Trans. on Computers*, Vol. C-27, No. 9, pp. 875-879, Sep. 1978.

[Pavlidis72] T. Pavlidis, "Segmentation of Pictures and Maps through Functional Approximation," *Computer Graphics and Image Processing*, Vol. 1, pp. 360-372, 1972.

[Pavlidis75] T. Pavlidis and S. Tanimoto, "Texture Identification by a Directed Split-and-Merge Procedure," *Proc. Conf. on Computer Graphics, Pattern Recognition, and Data Structures*, Los Angeles, pp. 201-203, May 1975.

[Pavlidis78] T. Pavlidis, "A Review of Algorithms for Shape Analysis," *Computer Graphics and Image Processing*, Vol. 7, pp. 243-258, 1978.

[Peleg78] S. Peleg, "Iterative Histogram Modification, 2", *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-8, No. 7, pp. 555-556, July 1978.

[Peleg80] S. Peleg, "A New Probabilistic Relaxation Scheme," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-2, No. 4, pp. 362-369, July 1980.

[Perkins80] W.A. Perkins, "Area Segmentation of Images Using Edge Points," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-2, No. 1, pp. 8-15, Jan. 1980.

[Pietikäinen82] M. Pietikäinen, A. Rosenfeld, and I. Walter, "Split-and-Link Algorithms for Image Segmentation," *Pattern Recognition*, Vol. 15, No. 4, pp. 287-298, 1982.

[Pingle 71] K. Pingle and J. Tenenbaum, "An Accommodating Edge Follower," *Proc. 2nd Int. Jnt. Conf. on Artificial Intelligence*, London, pp. 1-7, Sep. 1971.

[Postaire81] J.G. Postaire and C.P.A. Vasseur, "An Approximate Solution to Normal Mixture Identification with Application to Unsupervised Pattern Classification," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-3, No. 2, pp. 163-179, Mar. 1981.

[Prager80] J.M. Prager, "Extracting and Labeling Boundary Segments in Natural Scenes," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-2, No. 1, pp. 16-27, Jan. 1980.

[Pratt78] W.K. Pratt, *Digital Image Processing*. New York: Wiley-Interscience, 1978.

[Preparata72] F.P. Preparata and S.R. Ray, "An Approach to Artificial Non-Symbolic Cognition," *Inf. Sci.*, Vol. 4, pp. 65-86, 1972.

[Prewitt66] J.M.S. Prewitt and M.L. Mendhelson, "The Analysis of Cell Images," *Ann. N.Y. Acad. of Sci.*, Vol. 128, pp. 1035-1053, 1966.

[Prewitt70] J.M.S. Prewitt, "Object Enhancement and Extraction." In B. Lipkin and A. Rosenfeld (eds.), *Picture Processing and Psychopictorics*, Academic Press, New York, pp. 75-149, 1970.

[Price76] K.E. Price, *Change Detection and Analysis in Multi-Spectral Images*, Ph.D. Thesis, Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, PA, Dec. 1976.

[Price78a] K.E. Price, "Matching Segments of Images," *Semiannual Technical Report*, Report 800, Image Processing Inst., Univ. of Southern California, pp. 2-22, Mar. 1978.

[Price78b] K.E. Price, "Symbolic Matching and Analysis with Substantial Changes in Orientation," *Semiannual Technical Report*, Report 800, Image Processing Inst., Univ. of Southern California, pp. 22-41, Mar. 1978.

[Price79] K.E. Price, "Segmentation," *Proc. IEEE Conf. on Pattern Rec. and Image Processing*, Chicago, pp. 512-514, Aug. 1979.

[Price81] K.E. Price, "Relaxation Matching applied to Aerial Images," *Proc. Image Understanding Workshop*, pp. 22-25, Apr. 1981.

[Ranade80] S. Ranade and J.M.S. Prewitt, "A Comparison of Some Segmentation Algorithms for Cytology," *Proc. 5th Int. Jnt. Conf. on Pattern Recognition*, Miami Beach, FL, pp. 561-564, Dec. 1980.

[Ratkovic79a] J.A. Ratkovic, *Performance Considerations for Image Matching Systems*, The Rand Corp., Santa Monica, CA, Report RAND/N-1217-AF, December 1979, 79 pp.

[Ratkovic79b] J.A. Ratkovic, *Structuring the Components of the Image Matching Problem*, The Rand Corp., Santa Monica, CA, Report RAND/N-1216-AF, December 1979, 35 pp.

[Ratkovic79c] J.A. Ratkovic, "Hybrid Correlation Algorithms – A Bridge between Feature Matching and Image Correlation," *Proc. 18th IEEE Conf. on Decision and Control*, Fort Lauderdale, FL, Vol. 2, pp. 1046-52, December 1979.

[Read72] J.S. Read and S.N. Jayaramamurthy, "Automatic Generation of Texture Feature Detectors," *IEEE Trans. on Computers*, Vol. C-21, No. 7, pp. 803-812, July 1972.

[Riseman77] E.M. Riseman and M.A. Arbib, "Computational Techniques in the Visual Segmentation of Static Scenes," *Computer Graphics and Image Processing*, Vol. 6, No. 3, pp. 221-276, June 1977.

[Robertson73] T.V. Robertson, P.H. Swain, and K.S. Fu, *Multispectral Image Partitioning*, TR-EE 73-26 (LARS Information Note 071373), School of Electrical Engineering, Purdue Univ., Aug. 1973.

[Robinson77] G.S. Robinson, "Color Edge Detection," *Optical Engineering*, Sep./Oct. 1977. Also *Proc. SPIE Symp. on Advances in Image Transmission Techniques*, San Diego, Vol. 87, Aug. 1976.

[Rosenfeld76a] A. Rosenfeld and A.C. Kak, *Digital Image Processing*, Academic Press, NY, 1976.

[Rosenfeld76b] A. Rosenfeld, R.A. Hummel, and S.W. Zucker, "Scene Labeling by Relaxation Operations," *IEEE Systems, Man, and Cybernetics*, Vol. SMC-6, No. 6, pp 420-433, June 1976.

[Rosenfeld78] A. Rosenfeld and L.S. Davis, "Iterative Histogram Modification," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. 8, pp. 300-302, 1978.

[Rosenfeld79] A. Rosenfeld and L.S. Davis, "Image Segmentation and Image Models," *Proc. IEEE*, Vol. 67, No. 5, pp. 764-772, May 1979.

[Rosenfeld80] A. Rosenfeld, C.Y. Wang, and A.Y. Wu, *Multispectral Texture*, Report TR-988 (and AFOSR-77-3271), Computer Science Ctr., Univ. of Maryland, College Park, Dec. 1980.

[Rubin80] S.M. Rubin, "Natural Scene Recognition Using LOCUS Search," *Computer Graphics and Image Processing*, Vol. 13, pp. 298-333, 1980.

[Rutkowski81] W.S. Rutkowski, S. Peleg, and A. Rosenfeld, "Shape Segmentation Using Relaxation," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-3, No. 4, pp. 368-375, July 1981.

[Sakai76] T. Sakai, T. Kanade, and Y. Ohta, "Model Based Interpretation of Outdoor Scenes," *Proc. 3rd Int. Jnt. Conf. on Pattern Recognition*, Coronado, CA, pp. 581-585, Nov. 1976.

[Samet79] H. Samet, "Quadtree Structures for Region Processing," *Proc. Image Understanding Workshop*, Los Angeles, pp. 36-41, Nov. 1979.

[Sarabi81] A. Sarabi and J.K. Aggarwal, "Segmentation of Chromatic Images," *Pattern Recognition*, Vol. 13, No. 6, pp. 417-427, 1981.

[Schachter75] B.J. Schachter, L.S. Davis, and A. Rosenfeld, *Scene Segmentation by Cluster Detection in Color Spaces*, Report TR-424, Univ. of Maryland Computer Science Dept., 22 pp., Nov. 1975.

[Schachter77] B.J. Schachter, L.S. Davis, and A. Rosenfeld, *Some Experiments in Image Segmentation by Clustering of Local Feature Values*, Report TR-510, Univ. of Maryland Computer Science Dept., Mar. 1977.

[Schachter79] B.J. Schachter, L.S. Davis, and A. Rosenfeld, "Some Experiments in Image Segmentation by Clustering of Local Feature Values," *Pattern Recognition*, Vol. 11, No. 1, pp. 19-28, 1979.

[Shafer80] S.A. Shafer, *MOOSE: User's Manual, Implementation Guide, Evaluation*, IFI-HH B-70/80, Fachbereich Informatik, Univ. of Hamburg, W. Germany, Apr. 1980.

[Shafer82] S. Shafer and T. Kanade, *Recursive Region Segmentation by Analysis of Histograms*, Proc. Int. Conf. on Acoustics, Speech, and Signal Processing, Paris, pp. 1166-1171, May 1982.

[Serreyn78] D. Serreyn and R. Larson, "Adaptive Threshold for an Image Recognition System - Background Results and Conclusions," *Proc. Image Understanding Workshop*, pp. 133-138, May 1978.

[Smith80] D.R. Smith, *CMU Image Format and Paging System*, IUS Document CMU003, Computer Science Dept., Carnegie-Mellon Univ., Nov. 1980.

[Sloan75] K.R. Sloan and R. Bajcsy, "A Computational Structure for Color Perception," *Proc. ACM '75*, Minneapolis, 1975.

[Sloan81] K.R. Sloan, Jr., "Dynamically Quantized Pyramids," *Proc. 7th Int. Jnt. Conf. on Artificial Intelligence*, Vol. 2, pp. 734-736, 1981.

[Somerville76] C. Somerville and J.L. Mundy, "One Pass Contouring of Images Through Planar Approximation," *Proc. 3rd Jnt. Conf. on Pattern Recognition*, Coronado, CA, pp. 745-748, Nov. 1976.

[Tanimoto75] S.L. Tanimoto and T. Pavlidis, "A Hierarchical Data Structure for Picture Processing," *Computer Graphics and Image Processing*, Vol. 4, pp. 104-113, 1975.

[Tanimoto77] S.L. Tanimoto and T. Pavlidis, "The Editing of Picture Segmentations using Local Analysis of Graphs," *Comm. of the ACM*, Vol. 20, No. 4, pp. 223-229, Apr. 1977.

[Tanimoto78] S.L. Tanimoto, "An Optimal Algorithm for Computing Fourier Texture Descriptors," *IEEE Trans. on Computers*, Vol. C-27, pp. 81-84, January 1978.

[Tenenbaum73] J.M. Tenenbaum, *On Locating Objects by Their Distinguishing Features in Multisensory*
Images," TN 84, Artificial Intelligence Center, SRI International, Sep. 1973.

[Tenenbaum74] J.M. Tenenbaum, T.D. Garvey, S. Weyl, and H.C. Wolf, *An Interactive Facility for Scene Analysis Research*, Technical Note 87, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, CA, Jan. 1974.

[Tenenbaum75] J.M. Tenenbaum and S. Weyl, "A Region-Analysis Subsystem for Interactive Scene Analysis," *Proc. 4th Int. Conf. on Artificial Intelligence*, Tbilisi, USSR, pp. 682-687, Sep. 1975.

[Tenenbaum76a] J.M. Tenenbaum and H.G. Barrow, "Experiments in Interpretation-Guided Segmentation," *Jnt. Conf. on Pattern Recognition and Artificial Intelligence*, June 1976. Also in *Artificial Intelligence*, Vol. 8, pp. 241-274, 1976.

[Tenenbaum76b] J.M. Tenenbaum and H.G. Barrow, "IGS: A Paradigm for Integrating Image Segmentation and Interpretation," *Proc. 3rd Int. Jnt. Conf. on Pattern Recognition*, Coronado, CA, pp. 504-513, Nov. 1976.

[Tenenbaum80] J.M. Tenenbaum, *Application of Image Understanding to Cartography: A Survey* of Relevant Research and Opportunities," Technical Report, Artificial Intelligence Center, SRI International, Dec. 1980.

[Thompson77] W.B. Thompson, "Textural Boundary Analysis," *IEEE Trans. on Computers*, pp. 272-275, Mar. 1977.

[Thompson80] W.B. Thompson, "Combining Motion and Contrast for Segmentation," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-2, No. 6, pp. 543-549, Nov. 1980.

[Tisdale79] G.E. Tisdale, A.R. Helland, and J. Shipley, *Automatic Target Cueing (AUTO-Q) System Engineering Model Design*, Westinghouse Systems Development Division, Baltimore, MD, Report 80-0488, Sep. 1979.

[Tomita73] F. Tomita, M. Yachida, and S. Tsuji, "Detection of Homogeneous Regions by Structural Analysis," *Proc. 3rd Int. Jnt. Conf. on Artificial Intelligence*, pp. 564-571, Aug. 1973.

[Tomita82] F. Tomita, Y. Shirai, and S. Tsuji, "Description of Textures by a Structural Analysis," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-4, No. 2, pp. 183-191, Mar. 1982.

[Troy73] E.B. Troy, E.S. Deutsch, and A. Rosenfeld, "Gray-Level Manipulation Experiments for Texture Analysis," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-3, No. 1, pp. 91-98, Jan. 1973.

[Tsuji73] S. Tsuji and F. Tomita, "A Structural Analyzer for a Class of Textures," *Computer Graphics and Image Processing*, Vol. 2, No. 3/4, pp. 216-231, Dec. 1973.

[Uhr72] L. Uhr, "Layered Recognition Cone Networks that Preprocess, Classify and Describe," *IEEE Trans. on Computers*, Vol. 21, pp. 758-768, 1972.

[Underwood77] S.A. Underwood and J.K. Aggarwal, "Interactive Computer Analysis of Aerial Color Infrared Photographs," *Computer Graphics and Image Processing*, Vol. 6, pp. 1-24, 1977.

[Wacker69] A.G. Wacker, *A Cluster Approach to Finding Spatial Boundaries in Multispectral Imagery*, LARS Information Note 122969, Purdue Univ., West Lafayette, Indiana, 1969.

[Weszka74] J.S. Weszka, R.N. Nagel, and A. Rosenfeld, "A Threshold Selection Technique," *IEEE Trans. on Computers*, Vol. C-23, No. 12, pp. 1322-1326, Dec. 1974.

[Weszka78] J.S. Weszka and A. Rosenfeld, "Threshold Evaluation Techniques," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-8, No. 8, pp. 622-629, Aug. 1978.

[Winkler78] G. Winkler and K. Vattrodt, "Measures for Conspicuousness," *Computer Graphics and Image Processing*, Vol. 8, pp. 355-368, 1978.

[Wolfe70] J.H. Wolfe, "Pattern Clustering by Multivariate Mixture Analysis," *Behavioral Research*, Vol. 5, pp. 329-350, 1970.

[Yakimovsky73a] Y. Yakimovsky, *Scene Analysis using a Semantic Base for Region Growing*, Report STAN-CS-73-380, Stanford U. Computer Science Dept., June 1973.

[Yakimovsky73b] Y. Yakimovsky and J.A. Feldman, "A Semantic Based Decision Theory Region Analyzer," *Proc. 3rd Int. Jnt. Conf. on Artificial Intelligence*, pp. 580-588, Aug. 1973.

[Yakimovsky76] Y. Yakimovsky, "Boundary and Object Detection in Real World Images," *J. of the ACM*, Vol. 23, No. 4, pp. 599-618, Oct. 1976.

[Yan77] J.K. Yan and D.J. Sakrison, "Encoding of Images Based on a Two-Component Source Model," *IEEE Trans. on Communications*, Vol. COM-25, No. 11, pp. 1315-1322, Nov. 1977.

[Yasnoff77] W.A. Yasnoff, J.K. Mui, and J.W. Bacus, "Error Measures for Scene Segmentations," *Pattern Recognition*, Vol. 9, pp. 217-231, 1977.

[Yoo78] J.R. Yoo and T.S. Huang, *Image Segmentation by Unsupervised Clustering and its Applications*, Report TR-EE-78-19, Purdue Univ., West Lafayette, Indiana, 1978.

[Zucker75] S.W. Zucker, A. Rosenfeld, and L.S. Davis, "Picture Segmentation by Texture Discrimination," *IEEE Trans. on Computers*, Vol. C-24, No. 12, pp. 1228-1233, Dec. 1975.

[Zucker76a] S.W. Zucker, "Region Growing: Childhood and Adolescence," *Computer Graphics and Image Processing*, Vol. 5, No. 3, pp. 382-399, Sep. 1976.

[Zucker76b] S.W. Zucker, "Relaxation Labelling and the Reduction of Local Ambiguities," *Proc. 3rd Int. Jnt. Conf. on Pattern Recognition*, Coronado, CA, pp. 852-861, Nov. 1976.

[Zucker77] S.W. Zucker, R.A. Hummel, and A. Rosenfeld, "Application of Relaxation Labeling to Line and Curve Enhancement," *IEEE Trans. on Computers*, Vol. C-26, No. 4, pp. 394-403, Apr. 1977. Correction in Vol. C-26, No. 9, pp. 922-929, Sep. 1977.

[Zucker78] S. W. Zucker and J.L. Mohammed, "Analysis of Probabilistic Relaxation Labelling Processes," *Proc. IEEE Conf. on Pattern Rec. and Image Processing*, Chicago, pp. 307-312, 1978.